IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

SPECIAL SECTION ON SUSTAINABLE INFRASTRUCTURES, PROTOCOLS, AND RESEARCH CHALLENGES FOR FOG COMPUTING

Received October 23, 2018, accepted November 19, 2018, date of publication November 23, 2018,
date of current version December 27, 2018.

Digital Object Identifier 10.1109/ACCESS.2018.2883105

HEAP: An Efficient and Fault-Tolerant Authentication and Key Exchange Protocol for Hadoop-Assisted Big Data Platform

DURBADAL CHATTARAJ[1], (Student Member, IEEE), MONALISA SARMA[1], ASHOK KUMAR DAS[2], (Senior Member, IEEE), NEERAJ KUMAR[3], (Senior Member, IEEE), JOEL J. P. C. RODRIGUES[4,5,6], (Senior Member, IEEE), AND YOUNGHO PARK[7], (Member, IEEE)

[1]Subir Chowdhury School of Quality and Reliability, IIT Kharagpur, Kharagpur 721 302, India
[2]Center for Security, Theory and Algorithmic Research, International Institute of Information Technology, Hyderabad 500 032, India
[3]Department of Computer Science and Engineering, Thapar University, Patiala 147 004, India
[4]National Institute of Telecommunications, Santa Rita do Sapucaí 37540-000, Brazil
[5]Instituto de Telecomunicações, 1049-001 Aveiro, Portugal
[6]University of Fortaleza, Fortaleza 60811-905, Brazil
[7]School of Electronics Engineering, Kyungpook National University, Daegu 41566, South Korea

Corresponding author: Youngho Park (parkyh@knu.ac.kr)

This work was supported in part by the Basic Science Research Program through the National Research Foundation of Korea funded by the Ministry of Science, ICT & Future Planning under Grant 2017R1A2B1002147, in part by the BK21 Plus Project funded by the Ministry of Education, South Korea, under Grant 21A20131600011, in part by Finep/Funttel through the Radiocommunication Reference Center project of the National Institute of Telecommunications, Brazil, under Grant 01.14.0231.00, in part by the National Funding from the Fundação para a Ciência e a Tecnologia under Project UID/EEA/500008/2013, and in part by the Brazilian National Council for Research and Development (CNPq) under Grant 309335/2017-5. This work was also supported by the Ministry of Human Resource Development, Government of India (to carry out this research work at the Subir Chowdhury School of Quality and Reliability, IIT Kharagpur), through the Institute Fellowship.

ABSTRACT Hadoop framework has been evolved to manage big data in cloud. Hadoop distributed file system and MapReduce, the vital components of this framework, provide scalable and fault-tolerant big data storage and processing services at a lower cost. However, Hadoop does not provide any robust authentication mechanism for principals' authentication. In fact, the existing state-of-the-art authentication protocols are vulnerable to various security threats, such as man-in-the-middle, replay, password guessing, stolen-verifier, privileged-insider, identity compromization, impersonation, denial-of-service, online/off-line dictionary, chosen plaintext, workstation compromization, and server-side compromisation attacks. Beside these threats, the state-of-the-art mechanisms lack to address the server-side data integrity and confidentiality issues. In addition to this, most of the existing authentication protocols follow a single-server-based user authentication strategy, which, in fact, originates single point of failure and single point of vulnerability issues. To address these limitations, in this paper, we propose a fault-tolerant authentication protocol suitable for the Hadoop framework, which is called the efficient authentication protocol for Hadoop (HEAP). HEAP alleviates the major issues of the existing state-of-the-art authentication mechanisms, namely operating-system-based authentication, password-based approach, and delegated token-based schemes, respectively, which are presently deployed in Hadoop. HEAP follows two-server-based authentication mechanism. HEAP authenticates the principal based on digital signature generation and verification strategy utilizing both advanced encryption standard and elliptic curve cryptography. The security analysis using both the formal security using the broadly accepted real-or-random (ROR) model and the informal (non-mathematical) security shows that HEAP protects several well-known attacks. In addition, the formal security verification using the widely used automated validation of Internet security protocols and applications ensures that HEAP is resilient against replay and man-in-the-middle attacks. Finally, the performance study contemplates that the overheads incurred in HEAP is reasonable and is also comparable to that of other existing state-of-the-art authentication protocols. High security along with comparable overheads makes HEAP to be robust and practical for a secure access to the big data storage and processing services.

INDEX TERMS Cloud computing, authentication, key agreement, big data security, hadoop, formal security, AVISPA.

I. INTRODUCTION

The existence of the digital universe is expanding by a factor of 300, from 130 exabytes to 40,000 exabytes, or 40 trillion gigabytes (more than 5,200 gigabytes for every man, woman, and child in 2020) from 2005 to 2020. From the recent time until 2020, the digital universe will about double every two years.[1] This of course, drag the attention of many researchers and practitioners in the field of Big Data storage and processing issue. To deal with this, various distributed file systems' namely Hadoop Distributed File System (HDFS) [1], Google File System (GFS) [2], MooseFS,[2] zFS [3], Ceph [4], etc. have evolved. However, among these, due to popularity, simplicity and easy availability (open source), HDFS (the principal component of Hadoop framework [3]) is widely used in industries and became the de facto standard platform for Big Data storage. HDFS has been evolved to provide the storage service, where data is reliably kept in a distributed fashion into different servers. In this connection, client stores Big Data into the geographically dispersed remote third party servers through an insecure channel. This excavates several security concerns as the storage service access to be made over an insecure communication channel. Towards the solution, a robust authentication mechanism is the preferred solution. In this synergy, different authentication protocols have been proposed such as Kerberos,[4] OAuth,[5] OpenID connect,[6] SAML,[7] etc. But for the sake of simplicity, scalability and applicability, operating system based security (i.e., password-based approach), Kerberos authentication protocol (i.e., password with possession-based approach) and delegated token based approaches are currently employed in Hadoop for enhancing its security [5]–[9].

To access Big Data storage or processing services over the Internet, it is necessary for an end user (or service server) to initially enroll himself (or itself) with the Key Distribution Center (KDC) or a Centralized Registration Authority (CRA) offline. In centralized registration mechanism, it is difficult to update secret credentials of user and Hadoop clusters vis-a-vis service servers dynamically. After enrollment, the end user can access Big Data storage or processing services from the service server remotely over the Web. Usually, in such a setting, the KDC (or CRA) stores the secret information of all the principals' in its database, where a single point of vulnerability and single point of failure makes the whole system jeopardized [10]. In order to address these issues, many schemes have been reported in [5] and [11]–[30] that are based on different techniques namely, smart card based approaches [18], [20], [27]–[29], one-time padding [13], [14], PKI (Public-Key Infrastructure) based approach [24], [31], implementation of a Trusted Computing Platform (TCP) [15], combination of both password and possession based strategy [25], [26], [30], authorization delegation based approach [22], [32], combined public and private key cryptography with random number generator based scheme [12], utilizing basic geometry structure based password storing [33] and Identity-Based Authentication (IBA) scheme [23], respectively. However, these explications are either expensive in terms of extra hardware cost or computationally intensive. Further, the existing approaches [24]–[26], [30] enrol an end user (or service server) by asking his *username* and *password* (or *service server identity* and *secret key*), where the username (or service server identity) is used as the primary credential, which is verified at the time of mutual authentication between user and service server respectively. In fact, selecting a username (or service server identity) is not enough to be considered as a strong private identifier. As a result, an adversary can easily incorporate different attacks, such as impersonation attacks and identity compromisation attacks by sniffing the username (or service server identity) from the insecure media [10]. Moreover, these approaches are not considered the user-side and service server-side identity untraceability and anonymity properties. In spite of this, the existing password-based user enrollment strategy [30] which is currently incorporated in Hadoop is vulnerable to password guessing, online or offline dictionary and stolen-verifier attacks. Additionally, the existing approach [30] derive client's secret key as the hash value of its password. Therefore, the key will remain same until client changes the current password. However, changing this password needs updating the enrolled data maintained by the KDC (or CRA) and this, in fact, invites many key rollover problems [10]. In addition to this, man-in-the-middle, privileged-insider, denial-of-service, workstation compromisation, chosen plaintext and replay attacks are the key security threats that are not properly addressed in the existing schemes [10].

In order to ensure mutual authentication and session key distribution between end user and service server (Namenode or JobTracker), in possession based (also called token based) approach, a trusted server distributes a token with large numbers of authentication parameters, that is, more parameters are included into the constitution of an Authentication Token (AT) and authorization token (or Service Token (ST)). Hence, AT and ST verification increases the overhead to the existing authorization server (or service server). In addition to this, tokens and session keys are stored into user's credential cache [24]–[26], [30] in the respective workstation, and each token has its own lifetime. So, it leads to workstation compromisation attack, disclosure of session key as well as

[1] http://www.emc.com/leadership/digital-universe/2012iview/executive-summary-a-universe-of.htm

[2] MooseFS: Can Petabyte Storage be super efficient. https://moosefs.com/

[3] Apache Hadoop: https://hadoop.apache.org/

[4] http://web.mit.edu/kerberos/

[5] http://www.oauth.net/

[6] http://openid.net/

[7] saml.xml.org/

misuse of tokens. Moreover, an end user blindly accepts the authentication services, that is, he completely rely on the trusted third party server (KDC's *AS*) issued shared secret session key without verifying the strong authenticity of the *AS*. Therefore, if the *AS* is compromised by a malicious insider, a byzantine attack can be induced into the system which can falsify the primitive operations and it can also lead to the wrong desires [10]. Nonetheless, some mutual authentication schemes [24]–[26], [30] use time synchronization for joint authentication between end user and the service servers. More precisely, all principals in a realm must be synchronized with a centralized time server. In fact, this is an overhead for the implementation of the protocol. In addition to this, clock in a distributed system may not be always synchronized, so it may cause a replay attack for both the end user and the service server [34]. In spite of this, in the existing authentication approaches [24]–[26], [30], a user blindly trusts the authentication server (*AS*) without verifying any cross parameters (e.g., message authentication codes, server-side generated one-way hash chain based one-time identifiers [16], digital signatures [10], etc.) after receiving the authentication token. To the best of our knowledge, there is no solution to verify the originality of *AS* except the timestamps and visualization of password (or session key protected authentication or service token) [10]. Hence, this shortcoming opens a possibility of impersonation attacks [34], where a compromised principle can falsify the basic operations of the authentication system. In addition to this, in Hadoop, there is no provision to verify the data integrity and confidentiality after archiving end user's or organization's Big Data into HDFS. Since, the raw data blocks (constructed from the Big Data) are stored into various Datanodes as plaintext format, it is easy for an adversary to modify the content of the data blocks easily [5]–[9], [35]–[41]. Additionally, the result of the processed Big Data utilizing MapReduce framework is stored into end user's local file system, so anybody can read this content. To the best of our knowledge, there is no solution exists to mitigate these issues.

A. MOTIVATION

To address the aforesaid issues and challenges of the existing authentication schemes that assists security in Hadoop framework, we set the following objectives in the proposed scheme:

1) The proposed protocol should have the capability to enroll the Hadoop cluster's service server online (instead centralized deployment of service servers) with the authentication service provider by advocating the scalability issue.
2) The proposed protocol should prevent different well-known attacks, such as man-in-the-middle, replay, denial-of-service, privileged-insider, impersonation, identity compromisation, ciphertext-only, sever-spoofing and chosen plaintext attacks.
3) The proposed scheme should have a fault-tolerant and dependable authentication architecture to address the existing SOV and SOF issues.
4) In the proposed scheme, the authentication task for both Big Data technology provider and user should be more robust and user friendly advocating less usage of security credentials and hardwares (smart card, biometric scanner, smart mobile device, etc.).
5) The proposed protocol should disseminate securely the session key between two communicating parties.
6) The proposed scheme should provide a mechanism to read, write and process the user's Big Data securely in Hadoop cluster.
7) The proposed protocol should support user and service server anonymity by hiding their original identities from eavesdroppers and privileged-insiders.
8) The proposed scheme should have a provision to generate a fresh session key securely in each session to mitigate the workstation compromisation attack.
9) The proposed protocol should have the capability to store the dictionary of password securely at the server-side to mitigate the offline dictionary, password guessing and stolen-verifier attacks.
10) The proposed scheme should able to establish the session key between two communicating parties without timestamps utilization.

To fulfill the above objectives, a two-server based authentication framework has been introduced. This framework is structured in such a way that it mitigates the single point of failure (SOF) and single point of vulnerability (SOV) issues. Further, the proposed framework resists various well known security threats, such as, man-in-the-middle, replay, privileged-insider, Denial-of-Service (DoS), chosen plaintext, password guessing, identity compromisation, impersonation, stolen-verifier, server spoofing, offline dictionary and workstation compromisation attacks. According to the policy of the proposed framework a service provider can enrol any number of Hadoop clusters vis-a-vis service servers online with the Key Distribution Center (KDC). In this framework, the proposed KDC consists of three different servers. Among them, two are public servers (one server interacts with clients only and the other communicate with Big Data service providers) and the other is private server. Initially, clients and Big Data service providers enrol themselves offline with the private server. After offline registration, both client and service provider would eligible for online registration through the respective KDC's public server. As the private server is hidden from universal access, it ensures the server-side security. In the proposed framework, after online registration, the service providers are able to enroll his service servers (Namenode servers or Job Trackers) online with the KDC. So, the service server registration is simple and scalable in nature. Mean while, the clients' are able to communicate directly with the service servers' (Namenode Servers or Job Trackers) in a Hadoop cluster after establishing a secret key with the KDC's public server followed by a two-server based mutual authentication process (we call it as single sign-on). This single sign-on facility provides the access to any number of service servers by accomplishing only one time authentication

with the KDC. As a solution to the server spoofing and DoS attacks, we consider here a two-factor (password and authorization token) based authentication strategy. To preserve privacy, in our scheme, we make the original identity of users and service servers fully anonymous. To enhance the robustness and correctness of entity authentication, we propose a new digital signature based entity verification scheme utilizing both symmetric and asymmetric key cryptography. As a remedy of chosen plaintext attacks, we use stateless Cipher Block Chaining (CBC) mode of symmetric encryption and decryption strategy where a random nonce is utilized as an Initial Vector (*IV*). To establish a secure session between two communicating parties, we propose a new pair-wise session key agreement policy using elliptic curve cryptography. As a solution to the client-side workstation compromisation attacks, we store client's secret information indirectly into a private place (server-side) and later on it can be fetched by the legitimate user's only. Moreover, to check the integrity of data blocks which resides into various chunk servers or Datanodes in HDFS, a Hash-based Message Authentication Code (HMAC) based secure HDFS-read and HDFS-write operations has been introduced.

B. RESEARCH CONTRIBUTIONS

The major research contributions devised in this paper are listed below.

- We propose a new secure and scalable enrollment methodology to register a cluster of service servers with the trusted third party server by eliminating traditional in-house (centralized) service server registration policy.
- We then introduce a new fault tolerant authentication framework to provide dependable authentication services for remote clients.
- Next, we propose a new digital signature based mutual authentication policy, where each principal is able to verify the legitimacy of other intended principals along with the trusted third party on which both the principal rely on.
- We also introduce a new approach for security credentials distribution and replication policies in order to mitigate server-side single point of failure and single point of vulnerability issues.
- To distribute the session key securely between two intended principals, we then propose an elliptic curve cryptography based session key distribution policy by utilizing the concept of in-memory caching.
- In addition, we propose a mechanism to disseminate the session key between two communicating entities without compromising their identities.
- The extensive formal security inspection by utilizing *de facto* Real-Or-Random (ROR) model and the informal security analysis substantiate that the proposed protocol can address various well-known attacks against active and passive adversaries.
- The formal security verification using the widely-used AVISPA tool has been carried out for the proposed protocol, and the AVISPA simulation results assist that the proposed scheme is secure against man-in-the-middle and replay attacks.
- To enhance security, the proposed protocol has a facility to dynamically update user's password and service server's secret credentials online with the help of the authentication servers.
- Finally, the proposed scheme is user-friendly in nature and the user needs to remember only his/her identity and password to login into the proposed authentication system.

C. ROAD MAP OF THE PAPER

The remainder of the paper is structured as follows. We discuss the network model of HDFS in Section II. Section III presents the related work associated with the entity authentication in Hadoop. The necessary related mathematical preliminaries are discussed in Section IV, which are helpful for describing and analyzing the proposed protocol. In Section V, we demonstrate the proposed scheme. Section VI presents both formal security analysis using the widely-accepted Real-or-Random (ROR) model and informal security analysis of the proposed protocol. In Section VII, we simulate the proposed protocol under the broadly-used On-the-Fly Model Checker (OFMC) and SAT-based Model Checker (SATMC) backends by utilizing the AVISPA tool and summarize the attack traces. Section VIII presents the performance analysis of the proposed scheme. In Section IX, we elaborate few appealing features as the realizations of the proposed scheme. Finally, we conclude the paper in Section X.

II. NETWORK MODEL OF HDFS

Apache Hadoop[8] is an open source and provides a new way for storing and processing Big Data. It consists of two core components. The former one is File Store (FS) and later one is a Distributed Processing System (DPS). The FS is called as HDFS[9] and the DPS is termed as MapReduce.[10] HDFS is a distributed file system designed for storing very large files with streaming data access patterns, running on clusters of commodity hardware. Files are divided into blocks (default block size is 64 MB) and blocks are replicated and stored at different chunk servers (also called slave servers). The basic architecture of HDFS is shown in Figure 1.

Note here, we have shown only two Namenode servers (NSs) and one JobTracker (JT) in Figure 1, but in practical HDFS federation architecture[11] it cloud vary up to n number of such servers. Intuitively, the three noteworthy classifications of machine roles in a single Hadoop Cluster (HC) are client machine (i.e., HDFS Client), Master Node (MN) (i.e., combination of Namenode and Job Tracker) and Slave Nodes (SNs) (i.e., combination of Datanodes (DNs) and

[8]https://hadoop.apache.org/releases.html
[9]https://hortonworks.com/apache/hdfs/
[10]https://hortonworks.com/apache/mapreduce/
[11]https://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/Federation.html

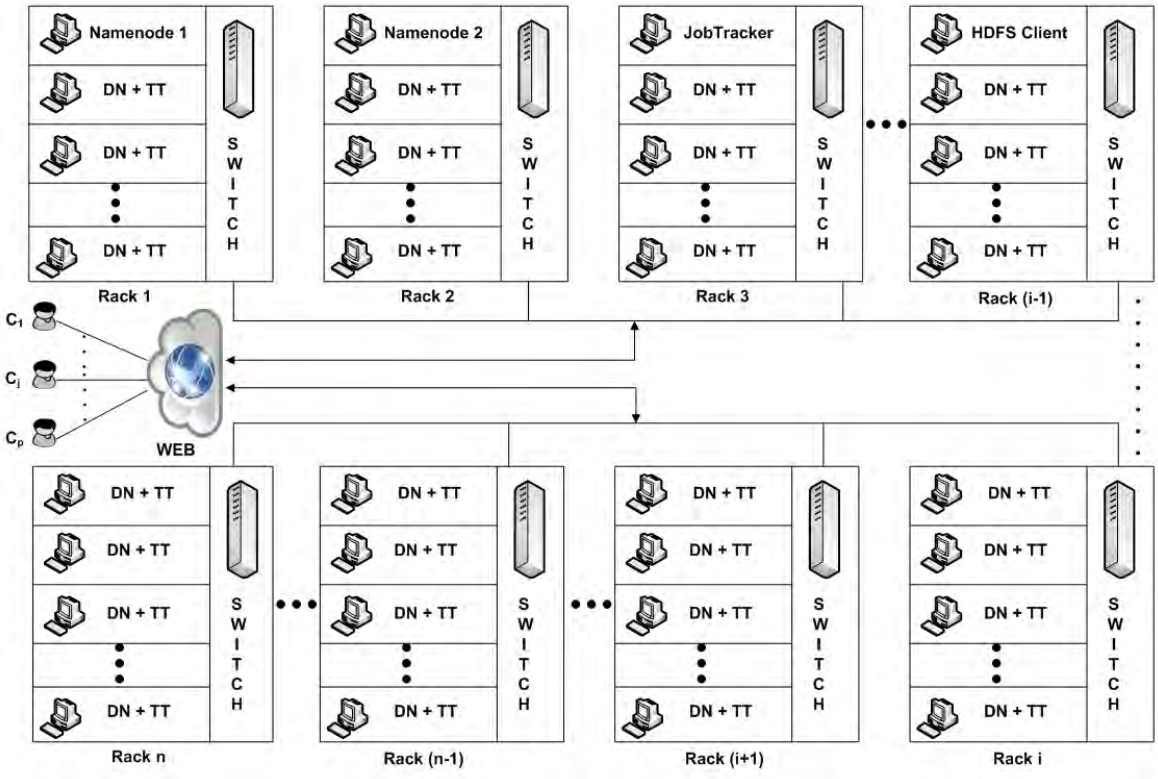

FIGURE 1. System architecture of HDFS Federation (new release).

Task Trackers (TTs)) (see Figure 1). All these components are connected through a communication network. In this architecture, for a single Hadoop Cluster say HC_j, the MN regulates two useful functions i.e., reliable data storage using HDFS and parallel computations utilizing MapReduce framework. The Namenode manages and facilitates distributed data storage services wherein the JT administers the parallel processing of stored data utilizing MapReduce (MR). Moreover, SNs are responsible for streamline data blocks storing and running the parallel computations over the stored data blocks. Each SN runs both Datanode and TT daemon and receives instructions from their MN. The TT daemon is a slave to the JT and the Datanode daemon acts as a slave to the Namenode. Client machine has Hadoop installed with all the cluster settings, however, it is neither a master nor a slave. Rather, the role of the Client machine is to load data into the cluster, submit MapReduce jobs portraying how that data ought to be processed, and after that, retrieve or view the results of the job when it's completed.

III. RELATED WORK

Hadoop framework has been evolved to manage a massive volume of data in Cloud. However, Hadoop does not provide any robust authentication mechanism for principals' authentication [5]–[7], [30], [34]. Since very few literature is available in this domain, the related work that is illustrated here is two fold: first, we discuss the state of the art authentication protocols and its variant that are actually studied in Hadoop (Big Data) platform and finally, we present the recent development of Authenticated Key Exchange (AKE) protocols in the domain of Cloud Computing platform as well as two-server based Password-assisted Authenticated Key Exchange (PAKE) schemes.

A. STATE OF THE ART AKEs FOR HADOOP (BIG DATA) PLATFORM

A limited number of authentication and key exchange protocols [5], [11], [13]–[17], [22]–[24], [30], [32], [42] has been found in this category.

Shen *et al.* [15] have proposed a theoretical prototype system combined with trusted platform support service. In their scheme, they have used a Trusted Computing Platform (TCP) to resolve the process of authentication in Hadoop. In TCP, the users identity is preserved and it is encrypted with users personal key and this mechanism is integrated in the hardware such as the BIOS and TPM. So it is very hard to decipher a user identity. The TCP is based on the Trusted Platform Module (TPM). The TPM is used to safeguard the system from different kind of hardware and software attacks. Authors have also pointed out the limitations in their scheme: (i) the stored data in the Datanodes will be decrypted when being accessed and will re-encrypted with different key after being accessed, the performance of system will be reduced, (ii) in order to make the authentication system trusted, some information are need to be stored among Namenodes, Datanodes and users, and finally (iii) TCB needs to fulfill many requirements

of server-side and user-side, so it may be raised bottleneck situation in the system.

Kohl and Neuman [30] proposed Kerberos authentication protocol. In their proposed approach, a user first registers with the system to avail the services. In this scheme, all the messages (i.e., authentication messages, service messages) are first encrypted using a shared secret key between two parties, and then the two parties communicate with each other with encrypted forms of messages. It may be noted that in the Kerberos protocol, a password based approach with a token based strategy needs to be followed for principal authentication. According to the current practice, a user makes an authentication request to an authentication server (*AS*) by means of a plain text containing ''username'' [34]. In this context, an attacker can eavesdrop the ''username'' and later expose himself to the *AS* as a legitimate user. In other word, an attacker can easily determine from the transmitted message that which users are currently online. In this situation, an attacker has scope to make man-in-the-middle attacks and replay attacks [10]. Further, an eavesdropper can make identity compromisation and impersonation attacks by stealing the ''username'' if the channel is insecure [34], [43], [44]. Moreover, the *AS* issues an authentication ticket (AT) to an end user after verifying only its ''username'' without verifying user's password or other security credentials [10]. However, as ''username'' is not a confidential credential, there is an opportunity for an attacker to get multiple authentication tickets by simply sending a ''username'' to the *AS*. As a consequence, a cryptanalyst can decrypt the ciphertexts (i.e., ATs) using some knowledge about underlying user's password. Thus, this scheme is vulnerable to Ciphertext-only Attack (COA). To avert this challenges, a public key infrastructure based Kerberos namely PKINIT [24] is reported and deployed in Hadoop. But, it is not properly addresses the user and service server's privacy issues and other security threats [10].

Somu *et al.* [14] proposed an authentication scheme for Hadoop and it is based on the encryption mechanism using one-time pad key. A random key is used to encrypt the password for secure transmission between the two servers (Registration Server and Back-end Server). Authors has claimed that their protocol makes the Hadoop environment more secure as the new random key for encryption is generated for each login. They also claimed that their scheme reduces the possibility to decrypt the cipher stored into the server for an adversary as it involves the knowledge about the valid random key. Sarvabhatla *et al.* [13] illustrated that Nivethitha *et al.*'s scheme is vulnerable to offline password guessing attack and on success of it, an attacker can perform all major attacks on HDFS. They proposed a new authentication service for Hadoop framework which is light weight and resists all major attacks as compared to Nivethitha *et al.*'s scheme. The authors also did a comparative analysis between their proposed user authentication service versus Nivethitha *et al.* scheme and found out that their scheme requires less number of hash operations as compare to Nivethitha *et al.*'s scheme.

For users' job authorization in Hadoop, an hash-based (MD5 and SHA-1) delegated job token mechanism has been reported in [5]. OAuth [22], OpenID connect [32] and SAML [42] are the new evolving authorization delegation based approach and it has been prioritize over traditional Kerberos protocol for principals' authorization and single sign-on capability incorporated in Hadoop.

Rahul and GireeshKumar [12] proposed a novel authentication framework for Hadoop. Their framework uses cryptographic functions such as public key cryptography, private key cryptography, hash functions, and random number generator. In this framework, they define a new key for each client and authenticate all clients and services using this key. They claimed that their authentication framework offers user data protection, a new way of privilege separations, and basic security needs for data storing inside HDFS.

Sadasivam *et al.* [11] proposed a novel authentication protocol for Hadoop in cloud environment, where they have used the basic properties of a triangle and modified two server-based model to improve the security level of Hadoop clusters. In their scheme, they have interpreted and alienated the user given password using the authentication server and stored in multiple back-end servers along with the corresponding username.

Kang and Zhang [23] proposed an Identity-Based Authentication (IBA) scheme which is of short key size, identity-based, non-interactive. This scheme divides the sharing users into the very same domain and in this domain relies on the sharing global master key to exercise mutual authentication. Their IBA scheme can be enabled by an emerging cryptographic technique from the bilinear pairing (i.e., Weil and Tate pairing [45] and its security can be assured by the Bilinear Diffie-Hellman Problem (BDHP)). But the limitation of this scheme is, if the global master key is leaked, then the total system will be jeopardized.

Sharma and Navdeti [6] listed various security mechanisms inside Apache Hadoop Stack. According to the authors, most of the cases, the Kerberos approach is preferably used for delivering authentication services.

Srinivas *et al.* [17] proposed 2PBDC: a privacy-preserving Big Data collection scheme in cloud environment utilizing elliptic curve cryptography. The authors shows that 2PBDC offers a better trade-off among the security and functionality features, communication and computation overheads. Aujla *et al.* [16] proposed SecSVA: Secure Storage, Verification, and Auditing of Big Data in the Cloud Environment. The authors presented an attribute-based secure data deduplication framework for data storage on the cloud, Kerberos-based identity verification and authentication, and Merkle hash-tree-based trusted third-party auditing on cloud.

B. STATE OF THE ART TWO-SERVER BASED PAKES AND AKEs FOR CLOUD COMPUTING PLATFORM

Karla and Sood [46] proposed cookie-based authentication and key exchange protocol for cloud and IoT environment. Later, Kumari *et al.* [47] pointed out the security flaws

of Karla and Sood scheme. Yang *et al.* [48] proposed an authentication scheme in a cloud environment setting. However, Chen *et al.* [49] pointed out the security pitfalls in Yang *et al.*'s scheme [48] that it is vulnerable to insider and impersonation attacks. To withstand these security loopholes in Yang *et al.*'s scheme, Chen *et al.* then designed a dynamic ID-based authentication scheme for cloud computing environment, which is based on the elliptic curve cryptography (ECC). Wang *et al.* [50] reviewed Chen *et al.*'s scheme [49], and proved that their scheme is vulnerable to offline password guessing as well as impersonation attacks. In addition, it was found that Chen *et al.*'s scheme does not provide user anonymity and it also has clock synchronization problem.

Later, Hao *et al.* [51] presented a time-bound ticket-based mutual authentication scheme for cloud computing. The purpose of using the time bound tickets is to reduce the server's processing overhead. Unfortunately, Jaidhar [52] identified that Hao *et al.*'s scheme [51] is insecure against denial-of-service attack during the password change phase. Wazid *et al.* [20] also proposed a provably secure user authentication and key agreement scheme for cloud computing environment. Their scheme withstands the weaknesses of the existing schemes and it also supports extra functionality features, such as user anonymity, and efficient password and biometric update phase in multi-server environment.

Recently, Gope and Das [53] proposed an anonymous mutual authentication scheme for ubiquitous mobile cloud computing services, which allows a legitimate mobile cloud user to enjoy n-times all the ubiquitous services in a secure and efficient way, where the value of n may differ based on the principal he or she has paid for. In addition, Odelu *et al.* [21] reviewed Tsai-Lo's scheme [54] and pointed out that their scheme does not provide the session-key security and also strong user credentials' privacy. To remove the security weaknesses found in Tsai-Lo's scheme, Odelu *et al.* designed a provably secure authentication scheme for distributed mobile cloud computing services. In addition to this, various biometric and smartcard based multi-factor authentication protocols [55]–[63] are found in the recent literature for multi-server environment. In spite of these approaches, various two-server based PAKE schemes [64]–[68] are evolved to mitigate server-side dependability (by addressing single point of failure and single point of vulnerability) and security issues.

IV. MATHEMATICAL PRELIMINARIES

The proposed authentication protocol is based on both asymmetric and symmetric key cryptography. In this context, in this work, we use Elliptic Curve Cryptography (ECC) and stateless CBC (cipher block chaining) mode of the Advanced Encryption Standard (AES) for both public and private key cryptography, respectively. The cryptographic hardness property related to Elliptic Curve Decisional Diffie-Hellman Problem (ECDDHP), Elliptic Curve Discrete Logarithm Problem (ECDLP) and Indistinguishability of Encryption scheme under Chosen Plaintext Attack (IND-CPA) are briefly explained in the following subsections. The proposed scheme also utilizes the collision-resistant cryptographic one-way hash function. This section provides a brief discussion about the aforesaid mathematical preliminaries as follows.

A. INDISTINGUISHABILITY OF ENCRYPTION SCHEME UNDER CPA

The indistinguishability of encryption scheme under chosen plaintext attack (IND-CPA) [61] is mathematically explained as follow:

Definition 1 (IND-CPA Secure): Assume SGL or MEL be the single or multiple eavesdropper/s respectively, and $OL_{EK_1}, OL_{EK_2}, \cdots, OL_{EK_M}$ be M different independent encryption oracles related to $EK_1, EK_2, \cdots, EK_M$ encryption keys, respectively. The advantage functions of SGL and MEL respectively, are defined as $\mathcal{A}dv^{IND-CPA}_{\mathcal{E},SGL}(K) = |2 \cdot Prob[SGL \leftarrow OL_{EK_1}; (p_0, p_1 \leftarrow_R SGL); \mu \leftarrow_R \{0,1\}; \tau \leftarrow_R OL_{EK_1}(p_\mu) : SGL(\tau) = \mu] - 1|$, and $\mathcal{A}dv^{IND-CPA}_{\mathcal{E},MEL}(K) = |2 \cdot Prob[MEL \leftarrow OL_{EK_1}, OL_{EK_2}, \cdots, OL_{EK_M}; (p_0, p_1 \leftarrow_R MEL); \mu \leftarrow_R \{0,1\}; \tau_1 \leftarrow_R OL_{EK_1}(p_\mu), \tau_2 \leftarrow_R OL_{EK_2}(p_\mu), \cdots, \tau_M \leftarrow_R OL_{EK_M}(pt_\mu) : MEL(\tau_1, \tau_2, \cdots, \tau_M) = \mu] - 1|$. We can say a symmetric cipher $\mathcal{E}$ is IND-CPA secure for the single or multiple eavesdropper/s setting if $\mathcal{A}dv^{IND-CPA}_{\mathcal{E},SGL}(K)$ (or $\mathcal{A}dv^{IND-CPA}_{\mathcal{E},MEL}(K)$) is negligible for the given security parameter K of any probabilistic and polynomial time adversary SGL (or MEL).

From Definition 1, it is easy to proof that a deterministic encryption scheme is not IND-CPA secure [61]. Further, there exists five generic modes of symmetric encryption scheme in the literature, namely Electronic Codebook (ECB), Output Feedback (OFB), Cipher Block Chaining (CBC), Cipher Feedback (CFB) and Counter (CTR) respectively. From these aforesaid modes, both ECB and stateful CBC modes are not IND-CPA secure, particularly in stateful CBC mode the value of Initialization Vector (IV) remains constrained which is shared between the sender and receiver. But, in stateless CBC mode, the IV value is chosen randomly for each message block. Thus, we use AES with stateless CBC mode of encryption or decryption policy throughout this paper so that it becomes IND-CPA secure [61].

B. ONE-WAY HASH FUNCTION AND ITS PROPERTIES

A one-way hash function h: $\{0,1\}^* \rightarrow \{0,1\}^l$ takes a binary string of variable length input, say $x \in \{0,1\}^*$ and results a binary string $h(x) \in \{0,1\}^l$ as an output of fixed length, say l bits. The formal definition of $h(\cdot)$ is provided as follows [69] [10].

Definition 2 (Collision-Resistant One-Way Hash Function): If an adversary $\mathcal{A}$'s advantage in finding collision in hash outputs with the execution time t is denoted by $Adv^{HASH}_{\mathcal{A}}(t)$, it is defined by $Adv^{HASH}_{\mathcal{A}}(t) = Pr[(x,y) \leftarrow_R \mathcal{A}: x \neq y \text{ and } h(x) = h(y)]$, where $Pr[E]$ is the probability of an event E and $(x,y) \leftarrow_R \mathcal{A}$ means the pair (x,y) is randomly chosen by $\mathcal{A}$. By an (η, t)-adversary $\mathcal{A}$ attacking the collision

resistance of $h(\cdot)$, it indicates that the execution time of $\mathcal{A}$ is at most t and that $Adv_{\mathcal{A}}^{HASH}(t) \leq \eta$.

Examples of a one-way hash function include the Secure Hash Standard (SHA-1) hashing algorithm and the stronger SHA-256 hashing algorithm [70].

C. ELLIPTIC CURVE AND ITS PROPERTIES

Suppose $m, n \in Z_p$, where $Z_p = \{0, 1, \ldots, p-1\}$ and $p > 3$ is a prime [10]. A non-singular elliptic curve $y^2 = x^3 + mx + n$ over the finite field Z_p is the set $E_p(m, n)$ of solutions $(x, y) \in Z_p \times Z_p$ to the congruence

$$y^2 \equiv x^3 + mx + n \pmod{p},$$

where $m, n \in Z_p$ such that $4m^3 + 27n^2 \neq 0 \pmod{p}$, and a point at infinity or zero point $\mathcal{O}$.

Note that $4m^3 + 27n^2 \neq 0 \pmod{p}$ is a necessary and sufficient condition to ensure a non-singular solution for the Eq. $x^3 + mx + n = 0$ [71]. $4m^3 + 27n^2 = 0 \pmod{p}$ implies the elliptic curve is singular. Let $P = (x_P, y_P)$, $Q = (x_Q, y_Q) \in E_p(m, n)$. Then $x_Q = x_P$ and $y_Q = -y_P$ when $P + Q = \mathcal{O}$. Also, $P + \mathcal{O} = \mathcal{O} + P = P$, for all $P \in E_p(m, n)$. Hasse's theorem states that the number of points on $E_p(m, n)$, denoted as $\#E$, satisfies the following inequality [72]:

$$p + 1 - 2\sqrt{p} \leq \#E \leq p + 1 + 2\sqrt{p}.$$

In other words, there are about p points on an elliptic curve $E_p(m, n)$ over Z_p. Also, $E_p(m, n)$ forms a commutative or an abelian group under addition modulo p operation.

- **Elliptic curve point addition:** Let $P, Q \in E_p(m, n)$ be two points on the elliptic curve. Then, $R = (x_R, y_R) = P + Q$ is calculated as follows [72]:

$$x_R = (\lambda^2 - x_P - x_Q) \pmod{p},$$
$$y_R = (\lambda(x_P - x_R) - y_P) \pmod{p},$$
$$\text{where } \lambda = \begin{cases} \dfrac{y_Q - y_P}{x_Q - x_P} \pmod{p}, \text{ if } P \neq -Q \\ \dfrac{3{x_P}^2 + m}{2y_P} \pmod{p}, \text{ if } P = Q. \end{cases}$$

- **Elliptic curve point scalar multiplication:** In ECC, multiplication is done as repeated additions. For example, $5P = P + P + P + P + P$, where $P \in E_p(m, n)$.

Definition 3 (ECDLP Assumption): Given an elliptic curve $E_p(m, n)$ and two points $R, S \in E_p(m, n)$, find an integer x such that $S = x \cdot R$.

Definition 4 (ECDDHP Assumption): Given a point R on an elliptic curve $E_p(m, n)$ and two other points $x \cdot R$, $y \cdot R \in E_p(m, n)$, find $(x \cdot y) \cdot R$.

V. THE PROPOSED PROTOCOL

In this section, we discuss the proposed scheme in detail. We call the proposed scheme as HEAP (**E**fficient **A**uthentication **P**rotocol for **H**adoop). The system architecture of HEAP is shown in Figure 2.

A. SYSTEM MODEL

Six types of principals are involved in the proposed system model: 1) client (C), 2) Big Data Service Provider (BDSP) 3) Namenode Server (NS) or Job Tracker (JT), 4) Client Management Server (CMS), 5) Namenode Management Server (NMS) and 6) Enrolment Server (ES). Both CMS and NMS are the public servers in two-server model, whereas ES is the private server. CMS is reachable to C_i utilizing a client application instance say HCA_j, where $i \in \{1, 2, 3, \cdots, n\}$. NMS is reachable to $BDSP_j$'s administrator using a server application instance say HSA_k, where $j \in \{1, 2, 3, \cdots, m\}$. Both CMS and NMS are reachable to adversaries but, ES operates in the background and it is fully supervised internally by the respective system administrator only. Thus, ES is fully trusted principal in the network. To make the proposed system model fault-tolerant, we distribute C_i's secret credentials into two servers (NMS and ES) whereas disseminates $BDSP$'s administrators and their deployable service server's (NS's and JT's) private (secret) information into another pair of servers (CMS and ES).

Initially, $BDSP$'s administrator needs to register all the service servers (NS and JT) of his own Hadoop cluster with ES online. To do this, $BDSP$'s administrator first enrol himself with ES and go through an authenticated key agreement procedure utilizing both NMS and CMS server. C enrol himself with ES during registration phase, but the authenticated session key agreement task will be held by both CMS and NMS. To prove the legitimacy of client C and $BDSP$'s administrator, both need to give responses about three different challenges (specifically maintained by a two-step verification process utilizing user identity, password and digital signature) assisted by both the servers (CMS and NMS). The successful legitimacy checking provides a Big Data storage or processing service server ticket to C and service server enrolment access privilege to $BDSP$'s administrator. The provided service ticket will then give access to the NS or JT after accomplishment of a mutual authentication and session key establishment process. The application instance HCA_j will give access to the CMS for C_i including adversaries whereas HSA_k will provide access to NMS for $BDSP$'s administrator including attackers. But, it is not possible for an adversary to access both CMS and NMS together utilizing a single application say HCA_j or HSA_k.

B. ADVERSARY MODEL

Presently, we have found three widely used threat models in the literature such as, Dolev-Yao threat model (DY model) [73], Canetti and Krawczyk adversary model (CK-adversary model) [74], and Extended Canetti and Krawczyk threat model (eCK-adversary model) [75] to model active and passive adversaries. However, we adopt DY model and CK-adversary model to study the proposed protocol.

Under DY model [73], an insecure channel between two communicating parties has been modeled mathematically in

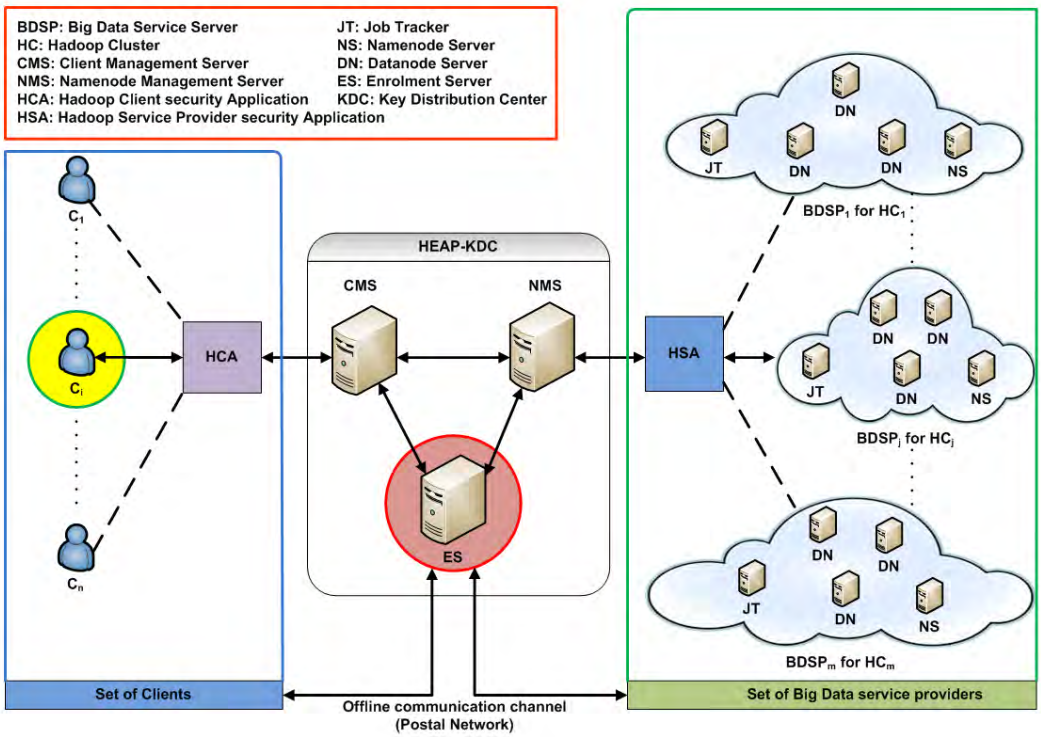

FIGURE 2. System architecture of HEAP.

such a way that an adversary $\mathcal{A}dv$ can intercept, delete or modify the exchanged messages. In addition to this, $\mathcal{A}dv$ may insert a fake message into the communication media to disgust the normal operations between two communicating parties. In the CK-adversary model [74] (the super set of DY model), the adversary $\mathcal{A}dv$ not only eavesdrop, delete or modify the exchanged messages between two communicating parties but also having the access to the session keys (short-term keys), long-term secret keys and session states of each party involves into the key agreement process. This model ensures the security of the authenticated key agreement protocol considering some sorts of security credentials (long-term and short-term) leakage and its impact on the security of other secret credentials.

We follow both DY and CK-adversary model in the proposed protocol, where we assume HEAP-KDC is trusted for both C and NS (or JT). Further, it is assumed that CMS and NMS are semi-trusted, whereas ES is fully trusted server. According to the policy of the DY model, any two parties such as C and NS or C and JT, are not considered as trustworthy principals in the network. Therefore, in this DY model, an adversary (active and passive) $\mathcal{A}dv$ can then eavesdrop, modify or delete the exchanged messages between C and NS or C and JT during communication. We also assume that the information stored at the C's workstation (mobile device) can be stolen by $\mathcal{A}dv$ and after obtaining the stolen information, $\mathcal{A}dv$ can perform the stolen-verifier and privileged-insider attacks. In addition, under the CK-adversary model, adversary $\mathcal{A}dv$ can have access some form of secret credentials including session key and session states between C and NS or C and JT. Under this assumption, the proposed protocol needs to show less security breech possibility of other entities' ($BDSP$, CMS, NMS and ES) secret credential due to the leakage of ''session ephemeral secrets'' between C and NS or C and JT.

However, in this study, we inspect several known security threats such as, chosen plain-text, denial-of-service, man-in-the-middle, online password guessing, server compromisation, replay, privileged-insider, stolen-verifier, offline password guessing, workstation compromisation, server spoofing and identity compromisation attacks considering both DY and CK-adversary model.

C. GENERAL OVERVIEW OF HEAP

HEAP goes through five basic operations: (i) HEAP-KDC configuration, (ii) user enrollment, (iii) Big Data service provider registration, (iv) Hadoop Cluster vis-a-vis service server enrollment and (v) mutual authentication and session key agreement between user and service server. Two security application instances namely HCA_j and HSA_k are running separately on user's workstation and service provider workstation to access a particular public server (CMS or NMS) of the HEAP-KDC's realm. More precisely, C accesses only CMS through the application HCA and the service provider accesses only NMS through the application HSA.

Initially, a Big Data Service Provider (*BDSP*) (or specifically the BDSP's administrator (*BA*)) needs to register himself with the *ES* through out-of-band channel (for example, a postal network) with his identity proof documents, service level agreement, service server details, etc. This entails to avail a one-time dummy identity, one-time dummy password and a pass-phrase to the service provider. After receiving these parameters offline, the service provider enrolls himself online with the HEAP-KDC utilizing both *NMS* and *HSA*. During online registration, the service provider needs to provide both the dummy identity and the dummy password to *NMS* through *HSA*. This entails to get a user account creation permission from *NMS*. In such a provision, the service provider sends his information namely, masked identity, masked password, email identity and mobile number, etc. to both *CMS* and *ES* servers utilizing *NMS* via a secure channel (utilizing pass-phrase as a key). After sending the service provider's information to *CMS* and *ES*, *NMS* only keeps the masked identity in its database. Similarly, an end user *C* register himself with *ES* by taking the help of *CMS* and *HCA*. During this operation, *C* sends his transformed identities, masked password, email identity and mobile number securely to *NMS* and *ES* servers via *CMS*. In this way, both user and service provider registration process has been accomplished.

After accomplishment of registration task, the *BDSP* needs to keep his original identity and password with himself. These two secrets are utilized at the time of service provider login phase. After successfully logged in into its workstation (locally) using *HSA*, the *BDSP* can register his Hadoop Cluster vis-a-vis Big Data storage and processing service servers (*NS*s' and *JT*s') with HEAP-KDC followed by a mutual authentication and key agreement process utilizing both *NMS* and *CMS* servers (two-server based authentication). This process is scalable in nature, where any Big Data service providers can able to enroll his cluster's service servers' online with HEAP-KDC. *BDSP* securely enrolls each service server to both *CMS* and *ES* through *NMS* by assigning a service server's masked identity and a masked password.

Similarly, after registration, *C* needs to keep his user identity and password secret with himself for logging in into its workstation. At the time of login, *C* needs to authenticate himself in its workstation locally utilizing *HCA* by providing his identity and password. After that, *C* goes through a mutual authentication and key formation process utilizing both *CMS* and *NMS* servers (dual server based authentication). This entails a short-term key to *C*. Utilizing this short-term key, *C* establishes a secure session with *CMS*. We call this process as single sign-on of client *C*. After the single sign-on task, *CMS* provides two encrypted tickets: (1) client ticket and 2) service server (either Big Data storage service server or processing service server) ticket to *C*. Utilizing this two tickets, *C* establishes a secret session with a service server (either *NS* or *JT*) associated with a particular cluster.

During two-server based mutual authentication and key agreement phase, the service provider (*BA*) enters his original identity and password to *HSA*. *HSA* computes a masked identity of the given identity. *HSA* then construct a digital signature on the masked identity utilizing a random nonce and *BA*'s chosen private key. After that, *HSA* sends the masked identity and the digital signature to *NMS*. In the same way, *NMS* construct a digital signature by signing its original identity with its private key. *NMS* sends *BA*'s masked identity and *BA*'s digital signature along with *NMS*'s encrypted digital signature and *NMS*'s identity to *CMS*. After verifying the identities of both *BDSP* and *NMS*, *CMS* decrypt *NMS*'s signature. Thereafter, *CMS* modifies both the digital signatures utilizing the previously shared pass-phrases of both the parties (*BDSP* and *NMS*) and *CMS*'s private key. *CMS* encrypts the modified signatures using *BDSP*'s masked password and *NMS*'s shared key. *CMS* then sends both the encrypted signatures to *NMS*. *NMS* decrypts the respective signature and verifies the legitimacy of both *BDSP* and *CMS* utilizing the previously loaded security parameters and sends the other signature to *BDSP*. After receiving the modified signature, *BDSP* (or *BA*) checks the legitimacy of both *NMS* and *CMS*. Finally, using the random nonces and ECC, both *BDSP* and *NMS* establish a session key between themselves. In the same way, at the time of single sign-on process, *C* establishes a short-term key with *CMS*.

After establishment of a secure session with *NS* utilizing *HCA*, *C* outsources its Big Data in terms of raw data blocks and its replicas into several Datanodes or chunk servers under the supervision of *NS*. In such a provision, *HCA* supplies the session key to the corresponding HDFS Client (HDCL) to achieve secure and integrity-assisted HDFS-read and HDFS-write operations. Thus, it will protect the user's confidential Big Data from the third party interception.

To make the proposed authentication protocol fault tolerant in terms of security credentials' replications, we keep the service providers and service servers credentials (mainly transformed identities and masked passwords) under *CMS*'s custody whereas disseminate the *C*'s credentials to *NMS*. Mean while, all the security credentials information are replicated concurrently into *ES* server. In addition to this, to transform the service provider's identity and password, two random secrets (one secret generated by *NMS* and other produced by the *HSA*) are embedded with the service provider's original user identity and password first, and afterwards a cryptographic one-way hash function has been applied with themselves. Similarly, *C*'s original user identity and password are encapsulated with *CMS*'s chosen secret and *HCA*'s secret, respectively. Thus, the aforesaid mechanism leads to create a strong password for both the parties (service provider and *C*) as well as reduce the chance of both single point of failure and single point of vulnerability issues.

To discuss HEAP methodology, we use various notations throughout the paper. The notations and their descriptions are listed in Table 1. In addition to this, we make certain

TABLE 1. Notations and their meanings.

Notation	Description
$BDSP$	Big Data Service Provider
$BDSP_{ID}$	Original identity of $BDSP$
$BDSP_{PWD}$	Password of $BDSP$
C	Client
C_{ID}	Original identity of C
CMS	Client Management Server
$Cert_E$	Certificate of an entity E issued by TCCA
CMS_{ID}	Public identity of CMS
ES	Enrollment Server or Registration Server (RS)
ES_{ID}	Public identity of ES
$E(\mathcal{K}:[msg])$	A message msg is encrypted using the key $\mathcal{K}$
$E_p(m,n)$	An elliptic curve $y^2 \equiv x^3 + mx + n \pmod p$ on the finite field Z_p
E	A entity such as C, $BDSP$, CMS, NMS, NS or JT
$\mathcal{G}$	A generator point $\in E_p(m,n)$ of some order (say, q)
HCA_j	An application instance running on C_i's workstation to access the CMS server
HCA_{ID}	Public identity for the application HCA_j
HSA_k	An application instance running on $BDSP_j$'s workstation to access the NMS server
HCA_{ID}	Public identity for the application HCA_j
HC_j	j^{th} Hadoop cluster
$h(\cdot)$	One way hash function such as (SHA-1 or SHA-2)
$\mathbb{H}_1(\cdot)$	A one way hash function, where $\mathbb{H}_1: \{0,1\}^* \rightarrow \mathbb{P}$
$\mathbb{H}_2(\cdot)$	A one way hash function, where $\mathbb{H}_2: \mathbb{P} \rightarrow \{0,1\}^*$
JT	Job Tracker
JT_{ID}	Public identity of JT
KDC	Key Distribution Center
$K_{NMS,C}$	A pass-phrase shared among NMS, ES and C
$K_{CMS,NMS}$	A shared symmetric key between CMS and NMS
$K_{CMS,NS}$	A shared symmetric key between CMS and NS
$K_{CMS,JT}$	A shared symmetric key between CMS and JT
$K_{CMS,BDSP}$	A pass-phrase shared among CMS, ES and $BDSP$
NMS	Namenode Management Server
NS	Namenode Server
NS_{ID}	Public identity of NS
NMS_{ID}	Public identity of NMS
PWD	Password of C
$\mathbb{P}$	A set of all points on the elliptic curve $E_p(m,n)$
Q_{CMS}	Public key of CMS
Q_{NMS}	Public key of NMS
Q_C	Public key of C
Q_{NS}	Public key of NS
Q_{BDSP}	Public key of $BDSP$
q	160-bit prime
$R_{C_i}^{CMS}$	A random number issued by CMS to transform C_{ID} and PWD
R_{BDSP}^{NMS}	A random number generated by NMS to transform $BDSP_{ID}$ and $BDSP_{PWD}$
S_{CMS}	Private key of CMS
S_{NMS}	Private key of NMS
S_C	Private key of C
S_{NS}	Private key of NS
S_{BDSP}	Private key of $BDSP$
SID_{NS}^j	A shared pass-phrase between j^{th} NS and CMS
SID_{JT}^j	A shared pass-phrase between j^{th} JT and CMS
$SID_{NMS,CMS}$ or $SID_{CMS,NMS}$	A shared pass-phrase between CMS and NMS
TNS_{ID}	A masked identity of NS
TJT_{ID}	A masked identity of JT
TCCA	Trusted Central Certification Authority
$\mathcal{MTU}$	A masked identity of C
$\mathcal{MTU}'$	A masked identity of $BDSP$
$\mathbb{Z}_q$	A finite field consisting of $0, 1, \cdots, (q-1)$ elements

assumptions for HEAP. These assumptions are described as follows:

1) Two security application instances say HCA_i and HSA_k are running concurrently into two separate workstations (user's and service provider's workstations) which enable to access the public servers (i.e., *CMS* and *NMS*s) in a HEAP-KDC realm.
2) An administrative application instances say, "h-admin" takes the responsibility of initial credentials' (i.e., private keys, pass-phrases and public identities) configuration in the key distribution center (HEAP-KDC) (also see Figure 2).
3) A Trusted Central Certification Authority (TCCA) chooses a generator $\mathcal{G}$ on the elliptic curve $E_p(m, n)$ of order q, and selects two cryptographic hash functions say, $\mathbb{H}_1(\cdot)$ and $\mathbb{H}_2(\cdot)$. Further, the TCCA generates a certificate $Cert_E$ for an entity E. The entity E randomly chooses $s_E \in \mathbb{Z}_q^*$ as private key and computes the corresponding public key as $Q_E = s_E \cdot \mathcal{G}$.
4) *BDSP* enrolls (i.e., online registration) himself with *CMS* via *NMS* followed by an offline registration with *ES*. After registration, *BDSP* needs to login into the system. After login, *BDSP* registers his Hadoop cluster vis-a-vis the service servers (*NS*s' and *JT*'s) with HEAP-KDC. Note that at least one cluster needs to be deployed with the KDC before initiating client registration.
5) *C* enrolls (i.e., online registration) himself with *NMS* via *CMS* followed by an offline registration with *ES*. After registration, *C* needs to login into the system for accessing the service servers (*NS*'s or *JT*'s).
6) *C* and *NS* or *C* and *JT* are not considered as trusted entity. They should mutually verify their legitimacy with the help of both *CMS* and *NMS*. After verification, either *C* and *NS* or *C* and *JT* become trusted to each other.
7) *CMS* keeps masked identities of *C*s, $R_{C_i}^{CMS}$s and all the secret credentials related to the Hadoop cluster vis-a-vis the service servers information whereas *NMS* stores masked identities of *BDSP*s, R_{BDSP}^{NMS}s and all the secret credentials of clients (*C*s). *ES* is having all the secret information of *C*s', *BDSP*s' and service servers'. Note here, it is not permissible for HEAP-KDC's server (*CMS* or *NMS* or *ES*) to store the secret credentials (mainly identity, password) of any principals in a plaintext format.
8) *C* or *BDSP* goes through a two-server based mutual authentication to avail services from the service server or deploy a new cluster with HEAP-KDC.
9) Finally, *ES* does not available to any other entities (*C*s, *NS*s and *JT*s) except *NMS* and *CMS*.

D. DETAILED DESCRIPTION OF HEAP

This section illustrates the detailed description of the proposed protocol phases as follows.

1) HEAP-KDC CONFIGURATION

HEAP undergoes an initial configuration phase, where a System Administrator (SA) frames the Key Distribution Center (i.e., HEAP-KDC). In this phase, all public servers are

TABLE 2. Summary of pre-loaded credentials into HEAP-KDC after the execution of *h-admin*.

HEAP-KDC	CMS	$CMS_{ID}, NMS_{ID}, ES_{ID}, SID_{(CMS,NMS)}, K_{(CMS,ES)}, K_{(CMS,NMS)}$
	NMS	$CMS_{ID}, NMS_{ID}, ES_{ID}, SID_{(NMS,CMS)}, K_{(NMS,ES)}, K_{(CMS,NMS)}$
	ES	$CMS_{ID}, NMS_{ID}, ES_{ID}, K_{(CMS,ES)}, K_{(NMS,ES)}$
Client and Service provider application	HCA	CMS_{ID}, HCA_{ID}
	HSA	NMS_{ID}, HSA_{ID}
	HCA	CMS_{ID}, HCA_{ID}

pre-loaded with secret credentials administered by the SA. In this regard, an *admin* process, called *h-admin*, runs at the time of HEAP-KDC configuration under the supervision of the SA. This phase follows a public server registration process to register both *CMS* and *NMS* with *ES*. In contrast, *ES* loads the public identities of *CMS* and *NMS* namely CMS_{ID} and NMS_{ID} in its database and share its public identity with both the servers. In addition, the *h-admin* process assigns three long-term shared secret symmetric keys between *CMS* and *ES*, *NMS* and *ES*, and *CMS* and *NMS* as $K_{(CMS,ES)}$, $K_{(NMS,ES)}$ and $K_{(CMS,NMS)}$, respectively. Further, *h-admin* generates one pass-phrase say $SID_{(CMS,NMS)}$. Finally, *h-admin* loads all these security parameters into respective servers. After the completion of *h-admin* process execution, the known security parameters with the HEAP-KDC are summarized in Table 2.

Note that *h-admin* also assigns two public identities for both client application (*HCA*) and service provider application (*HSA*), say HCA_{ID} and HSA_{ID}, respectively. HCA_{ID} is publicly available to *CMS* and HSA_{ID} is publicly available to *NMS*. Similarly, the public identity of *CMS* is known to *HCA* whereas the public identity of *NMS* are known to *HSA*. These identities are also useful for the key agreement process at the beginning of the client (or Big Data service provider) registration and login phases, respectively.

2) BIG DATA SERVICE PROVIDER REGISTRATION

Suppose a Big Data Service Provider (*BDSP*) wants to deploy a new Hadoop Cluster (HC_j) for providing the data storage and processing services via Internet. To enrol the HC_j with HEAP-KDC, *BDSP*'s Administrator (*BA*) needs to register himself with *ES* offline (i.e., via the out-of-band channel or postal network) by giving the detail about total number of service servers', service types, Service Level Agreements (SLAs), service servers' location information, service servers' subscription, payment documents, etc. This entails the *BA* to avail a synthetic identity ($S_{BDSP_{ID}}$), a synthetic password ($S_{BDSP_{PWD}}$) and a pass-phrase ($K_{CMS,BDSP}$) respectively, via the out-of-band channel to the *BDSP*'s physical address. Before sending these three security credentials to *BA*, *ES* securely sends $S_{BDSP_{ID}}$ and $S_{BDSP_{PWD}}$ to the *NMS* for creating a synthetic account of *BA*. *ES* also sends securely the pass-phrase $K_{CMS,BDSP}$ to the *CMS*. Note that *BA* needs to use $S_{BDSP_{ID}}$, $S_{BDSP_{PWD}}$ and $K_{CMS,BDSP}$ only for once to create his own profile into *CMS* with the help of *NMS*. Although, *BA* is permissible to use this pass-phrase $K_{CMS,BDSP}$ for his password updation and service server registration process.

Figure 3 summarizes the *BA*'s registration phase, which contains the following steps:

Step BDSPRG1: *BA* enters $S_{BDSP_{ID}}$ and $S_{BDSP_{PWD}}$ into *HSA*. *HSA* generates a random nonce say n_1'. *HSA* encrypts $S_{BDSP_{ID}}$ and n_1' using $S_{BDSP_{PWD}}$ as $\mathcal{BDSP}_{Dtl} = E(S_{BDSP_{PWD}} : [S_{BDSP_{ID}}, n_1'])$. *HSA* sends the message $msg_{B1} = \{NMS_{ID}, n_1', \mathcal{BDSP}_{Dtl}\}$ to the *NMS*.

Step BDSPRG2: After receiving msg_{B1}, *NMS* decrypts $\mathcal{BDSP}_{Dtl}$ and checks the availability of $S_{BDSP_{ID}}$ in its database. If it exists then, *BA* is permissible to create its own account into *CMS* and return $msg_{B2} = \{NMS_{ID}, CMS_{ID}, n_1'\}$, and goto Step BDSPRG3 else, reject *BA*'s request.

Step BDSPRG3: *BA* enters its original identity as $BDSP_{ID}$ in *HSA*. *HSA* chooses a random secret d and transforms the $BDSP_{ID}$ as $\mathcal{TU}' = h(BDSP_{ID}|| \ d \ ||HSA_{ID})$. Using this transformed identity $\mathcal{TU}'$, both *BA* and *NMS* establishes a shared secret key ($SK_{(NMS,BDSP)}$) between themselves followed by a mutual authentication process utilizing the similar analogy say "Initial key establishment between C_i and *FEAS*" reported in DPTSAP [10].

Step BDSPRG4: *BA* enters its new password as $BDSP_{PWD}$ into *HSA*. *HSA* computes the masked password $BDSP^*_{PWD} = h(h(d|| \ BDSP_{PWD}) \ ||R^{NMS}_{BDSP})$. Note that at the time of shared secret key establishment process R^{NMS}_{BDSP} was generated by *NMS* and it has been delivered securely using the key $SK_{(NMS,BDSP)}$. Further, both *BA*'s request and *NMS*'s response messages using DPTSAP [10] are represented as msg_{B3} and msg_{B4}, respectively in Figure 3.

Step BDSPRG5: *HSA* encrypts $BDSP^*_{PWD}$ using $K_{(CMS,BDSP)}$ as $\mathcal{BDSP}'_{Dtl} = E(K_{(CMS,BDSP)} : [BDSP^*_{PWD}, \mathcal{MTU}'])$, where $\mathcal{MTU}' = h(BDSP_{ID}|| \ d|| \ HSA_{ID}|| \ R^{NMS}_{BDSP})$. *HSA* sends the message $msg_{B5} = \{NMS_{ID} \ , n_2', \mathcal{BDSP}'_{Dtl}\}$ to the *NMS*.

Step BDSPRG6: After receiving msg_{B5}, *NMS* broadcasts both $\mathcal{MTU}'$ and $\mathcal{BDSP}'_{Dtl}$ to both *ES* and *CMS* servers (see $msg_{B5.1}$). *NMS* keeps only $\mathcal{MTU}'$ and R^{NMS}_{BDSP} in its database and deletes other informations, wherein, after decrypting $\mathcal{BDSP}'_{Dtl}$, both *ES* and *CMS* stores $\mathcal{MTU}'$ and $BDSP^*_{PWD}$ into their corresponding databases (see $msg_{B5.2}$). Finally, *NMS* sends a registration confirmation message as msg_{B6} to *BA* through *HSA* and goto Step BDSPRG7.

Step BDSPRG7: *HSA* computes $d^* = d \oplus h(BDSP_{ID} \ ||BDSP_{PWD})$, $R^{NMS*}_{BDSP} = R^{NMS}_{BDSP} \oplus h(d \ ||BDSP_{PWD})$, $\mathcal{FI} = h(BDSP_{ID}|| \ BDSP_{PWD})$, $K^*_{(CMS,BDSP)} = K_{(CMS,BDSP)} \oplus h(BDSP_{PWD} \ || \ R^{NMS}_{BDSP})$ and $BDPW = h(BDSP_{ID} \ ||HSA_{ID} \ ||BDSP_{PWD}|| \ R^{NMS}_{BDSP} \ ||d)$, and then stores these information into *HSA*'s database. *HSA* will use these information at the time of *BA*'s login, service server registration and password updation phases.

Note that after accomplishment of the registration process, *BA* needs to remember only two parameters $BDSP_{ID}$ and $BDSP_{PWD}$ to login into the system and then he can enrol any

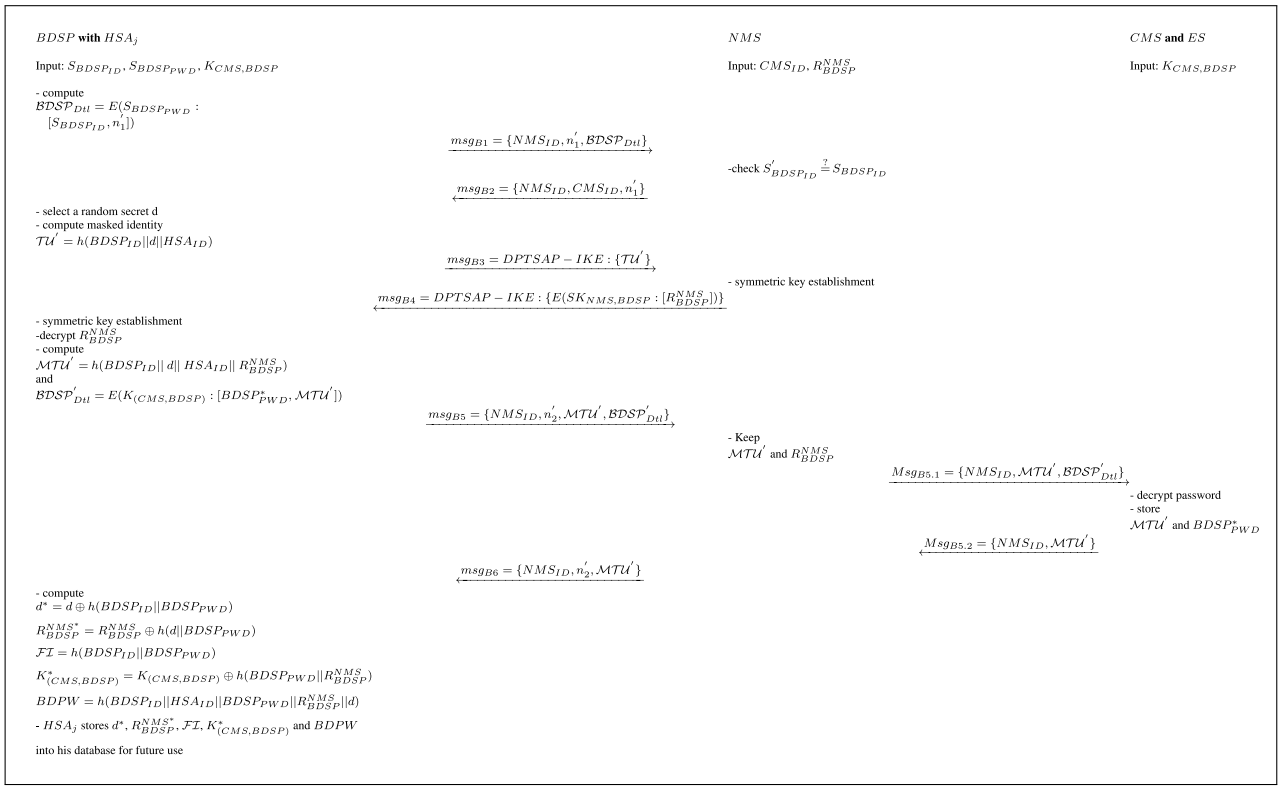

FIGURE 3. Summary of service provider registration.

number of Hadoop Clusters (*HC*) online with HEAP-KDC. Thus, the proposed service provider registration scheme is user friendly and scalable in nature. The online enrolment process of the *HC* driven by the *BA* is presented as follows.

3) HADOOP CLUSTER REGISTRATION

Hadoop cluster registration vis-a-vis service servers enrolment phase is proposed to carry out the different activities starting from service provider login to online registration of service servers (mainly, all Namenode servers and JobTrackers belongs to a particular Hadoop cluster say HC_j, where $j = 1, 2, \cdots, m$). The proposed enrolment phase consists of three activities: 1) service provider (*BA*) login, 2) mutual authentication and session key establishment between *BA* and *NMS* and 3) service servers registration. The detail steps involved in this process are discussed as follows. For simplicity, in this study, we assume a particular cluster say HC_j consists of a single Namenode server (NS_j) and a single Job Tracker server (JT_j), and NS_j is responsible to provide the Big Data storage services wherein JT_j yields the Big Data processing services to the remote user online.

a: SERVICE PROVIDER LOGIN AT WORKSTATION

After the completion of *BA*'s registration process, *BA* tries to login into the system using the server application *HSA*. Note here, before initiating service provider (*BA*) login into *BA*'s workstation, *HSA* loads all *BA*'s transform identities (that is, $\mathcal{FI}_i$, where $i = 1, 2, \cdots, k$) into its browser cookie. A diagram summarizing several communication message exchanges between *BA* and *NMS* involved throughout the service provider login, authenticated key establishment and service server registration process are shown in Figure 9, and it contains the following steps:

Step BDSPL1: *BA* enters his original identity $BDSP_{ID}$ and password $BDSP_{PWD}$ into *HSA*. *HSA* computes $\mathcal{FI}^{*} = h(BDSP_{ID}||BDSP_{PWD})$ and checks this entry exists in its cookie or not. If it exists then *HSA* loads the respective d^{*}, $R^{NMS^{*}}_{BDSP}$, $K^{*}_{CMS,BDSP}$ and $BDPW$ entries for the same *BA*.

Step BDSPL2: *HSA* computes $d = d^{*} \oplus h(BDSP_{ID}||BDSP_{PWD})$, $R^{NMS}_{BDSP} = R^{NMS^{*}}_{BDSP} \oplus h(d||BDSP_{PWD})$ and $K_{(CMS,BDSP)} = K^{*}_{(CMS,BDSP)} \oplus h(R^{NMS}_{BDSP}||BDSP_{PWD})$, and goto Step BDSPL3. Else, *BA* can repeat Step BDSPL1 with another user identity and password.

Step BDSPL3: *HSA* computes $BDPW^{*} = h(BDSP_{ID}||HSA_{ID}||BDSP_{PWD}||d)$ and checks if the condition $BDPW^{*} = BDPW$ holds or not. If it holds, *BA* is treated as an authentic service provider.

b: AUTHENTICATED KEY AGREEMENT PHASE

After successful logging in into the system, *HSA* initiate an authenticated key formation process between *BA*/*BDSP* and *NMS*. The detail steps involved in this process are shown in Figure 9 and are discussed as follows.

function bdspSignPairGen[ ]($\mathcal{MTU}'$, S_{BDSP})
{
Input: $BDSP/BA$'s masked identity and $BDSP$'s private key
Output: Two ECC points as λ^1_{BDSP} and λ^2_{BDSP}
Domain Parameters: $\mathbb{H}_2(\cdot), \mathbb{H}_1(\cdot), \mathcal{G}, q$
1. Select a pseudo-random number $R_{BDSP} \in \mathbb{Z}^*_q$
2. Compute $\lambda^1_{BDSP} := R_{BDSP} \cdot \mathcal{G}$
3. Compute λ^2_{BDSP}[c] $:= R_{BDSP} \cdot \mathbb{H}_2(\mathbb{H}_1(\mathcal{MTU}')) + S_{BDSP} \cdot \mathbb{H}_2(\lambda^1_{BDSP}) \pmod q$
4. return $R_{BDSP}, \lambda^1_{BDSP}, \lambda^2_{BDSP}$
}

[c]Note: $BDSP$ digitally sign on both $\mathcal{MTU}'$ and λ^1_{BDSP} using its private key S_{BDSP}.

FIGURE 4. ***BDSP*'s signature generation process.**

function nmsSignPairGen[ ](NMS_{ID}, S_{NMS})
{
Input: NMS's identity and NMS's private key
Output: Two ECC points μ^1_{NMS} and μ^2_{NMS}
Domain Parameters: $\mathbb{H}_2(\cdot), \mathbb{H}_1(\cdot), \mathcal{G}, q$
1. Choose a pseudo-random number $R_{NS} \in \mathbb{Z}^*_q$
2. Compute $\mu^1_{NMS} := R_{NMS} \cdot \mathcal{G}$
3. Compute μ^2_{NMS}[d] $:= R_{NMS} \cdot \mathbb{H}_2(\mathbb{H}_1(NMS_{ID})) + S_{NMS} \cdot \mathbb{H}_2(\mu^1_{NMS}) \pmod q$
4. return $R_{NMS}, \mu^1_{NMS}, \mu^2_{NMS}$
}

[d]Note: NMS digitally sign on both NMS_{ID} and μ^1_{NMS} using its private key S_{NMS}.

FIGURE 5. ***NMS*'s signature generation process.**

function cmodifiedSignPairGen[ ](λ^2_{BDSP}, $E(K_{CMS,NMS} : [NMS_{ID}, \mu^2_{NMS}])$)
{
Input: C_i's partial signature and encrypted NMS's partial signature
Output: Both C_i's and NMS's modified signature
Domain Parameters: $\mathbb{H}_2(\cdot), \mathbb{H}_1(\cdot), \mathcal{G}, q$
Data: $BDSP^*_{PWD}, K_{CMS,BDSP}, K_{CMS,NMS}, SID_{NMS,CMS}, NMS_{ID}, \mathcal{MTU}'$
if($\mathcal{MTU}'' = \mathcal{MTU}'$ && $NMS'_{ID} = NMS_{ID}$)
{
1. $\mu^{new}_{NMS} = \mu^2_{NMS} + S_{CMS} \cdot \mathbb{H}_2(\mathbb{H}_1(K_{CMS,BDSP})) \pmod q$
2. $\lambda^{new}_{BDSP} = \lambda^2_{BDSP} + S_{CMS} \cdot \mathbb{H}_2(\mathbb{H}_1(SID_{CMS,NMS})) \pmod q$
3. return $\mu^{new}_{NMS}, \lambda^{new}_{BDSP}, \mathcal{MTU}'$
}
else
{
4. return null // reject $BDSP$ and NMS //
}
}

FIGURE 6. **Signature updation process into *CMS*.**

Step MASKA1: *HSA* computes $BDSP_i$ masked identity $\mathcal{MTU}' = h(BDSP_{ID}||d||\ HSA_{ID}||\ R^{NMS}_{BDSP})$. Further, *HSA* generates two pseudo-random numbers (λ^1_{BDSP} and λ^2_{BDSP}) utilizing $\mathcal{MTU}'$ and different pre-loaded security domain parameters as the input to a function say bdspSignPairGen[ ](·) (see Figure 4).

Step MASKA2: *HSA* forms the message $M_1 = \{\mathcal{MTU}', \lambda^1_{BDSP}, \lambda^2_{BDSP}, NMS_{ID}, Cert_{BDSP}\}$ and sends it to *NMS* via a public channel.

Step MASKA3: After receiving M_1, *NMS* searches its database to check the existence of $\mathcal{MTU}'$. If it finds the same then *NMS* generates two pseudo-random numbers (μ^1_{NMS}, μ^2_{NMS}) by taking NMS_{ID} and other domain parameters as the input to a function say nmsSignPairGen[ ](·) (see Figure 5).

Step MASKA4: *NMS* constructs a message $M_2 = \{NMS_{ID}, CMS_{ID}, \lambda^2_{BDSP}, E(K_{CMS,NMS} : [\mathcal{MTU}', \mu^2_{NMS}])\}$ and sends it to *CMS* via a public channel.

Step MASKA5: After receiving the message M_2, *CMS* searches both $\mathcal{MTU}'$ and NMS_{ID} in its database. If both are exists then *CMS* understands that both *BDSP* and *NMS* are legitimate parties, and goto Step MASKA6; otherwise, rejects *NMS*'s request.

Step MASKA6: *CMS* loads both *BDSP*'s pass-phrase ($K_{CMS,BDSP}$) and *NMS*'s secret identity ($SID_{CMS,NMS}$) from its database. *CMS* modifies both *BDSP*'s and *NMS*'s partial signatures (i.e., μ^2_{NMS} and λ^2_{BDSP}) using a function cmodifiedSignPairGen[ ](·) (see Figure 6) as (1) μ^{new}_{NMS}[1] $= \mu^2_{NMS} + S_{CMS} \cdot \mathbb{H}_2(\mathbb{H}_1(K_{CMS,BDSP})) \pmod q = R_{NMS} \cdot \mathbb{H}_2(\mathbb{H}_1(NMS_{ID})) + S_{NMS} \cdot \mathbb{H}_2(\mu^1_{NMS}) + S_{CMS} \cdot \mathbb{H}_2(\mathbb{H}_1(K_{CMS,BDSP})) \pmod q$ and (2) λ^{new}_{BDSP}[2] $= \lambda^2_{BDSP} + S_{CMS} \cdot \mathbb{H}_2(\mathbb{H}_1(SID_{CMS,NMS})) \pmod q = R_{BDSP} \cdot \mathbb{H}_2(\mathbb{H}_1(\mathcal{MTU}')) + S_{BDSP} \cdot \mathbb{H}_2(\lambda^1_{BDSP}) + S_{CMS} \cdot \mathbb{H}_2(\mathbb{H}_1(SID_{CMS,NMS})) \pmod q$ and goto Step MASKA7.

Step MASKA7: *CMS* constructs a message $M_3 = \{CMS_{ID}, NMS_{ID}, Cert_{CMS}, E(K_{CMS,NMS} : [\lambda^{new}_{BDSP}]), E(BDSP^*_{PWD} : [\mathcal{MTU}', \mu^{new}_{NMS}])\}$ and sends the same to *NMS*.

Step MASKA8: *NMS* verifies the legitimacy of both *BDSP* and *CMS* utilizing the bcmsVerification(·) function shown in Figure 7. If the function bcmsVerification(·) returns "Accept", then *NMS* construct a session key $SK = R_{NMS} \cdot \lambda^1_{BDSP} = R_{NMS} \cdot R_{BDSP} \cdot \mathcal{G}$ and a message $M_4 = \{\mathcal{MTU}', NMS_{ID}, \mu^1_{NMS}, Cert_{CMS}, Cert_{NMS}, E(BDSP^*_{PWD} : [\mathcal{MTU}', \mu^{new}_{NMS}]\}$. *NMS* sends M_4 to *BDSP*, and goto Step MASKA9; otherwise, it rejects *BDSP*'s request.

Step MASKA9: After getting the message M_4, *BDSP* verifies both *NMS* and *CMS* utilizing the following function ncmsVerification(·) (see Figure 8). If the function ncmsVerification(·) returns "Accept" then *BSDP* receives *NMS*'s response and constructs a session key $SK_{BDSP,NMS} = R_{BDSP} \cdot \mu^1_{NMS} = R_{BDSP} \cdot R_{NMS} \cdot \mathcal{G}$ otherwise; rejects *NMS*'s response.

Proof of Correctness: In order to verify the legitimacy of both *BDSP* and *CMS*, *NMS* needs to check $\lambda^{new}_{BDSP} \cdot \mathcal{G} = \lambda^2_{BDSP} \cdot \mathcal{G} + Q_{CMS} \cdot \mathbb{H}_2(\mathbb{H}_1(SID_{NMS,CMS}))$. To satisfy the verification condition, it must holds $\lambda^2_{BDSP} \cdot \mathcal{G} = \lambda^1_{BDSP} \cdot \mathbb{H}_2(\mathbb{H}_1(\mathcal{MTU}')) + Q_{BDSP} \cdot \mathbb{H}_2(\lambda^1_{BDSP}) = R_{BDSP} \cdot \mathbb{H}_2(\mathbb{H}_1(\mathcal{MTU}')) \cdot \mathcal{G} + S_{BDSP} \cdot \mathbb{H}_2(\lambda^1_{BDSP}) \cdot \mathcal{G}$ and $Q_{CMS} \cdot \mathbb{H}_2(\mathbb{H}_1(SID_{NMS,CMS})) = S_{CMS} \cdot \mathbb{H}_2(\mathbb{H}_1(SID_{NMS,CMS})) \cdot \mathcal{G}$. Similarly, to verify the legitimacy of both *NMS* and *CMS*,

[1]Note: *CMS* digitally sign on the messages say μ^2_{NMS}, $K_{CMS,NMS}$ and μ^1_{NMS} using CMS's private key S_{CMS}.

[2]Note: *CMS* digitally sign on the messages say $\mathcal{MTU}'$, $SID_{CMS,NMS}$ and λ^1_{BDSP} using its private key S_{CMS}.

function bcmsVerif($E(K_{CMS,NMS} : [\lambda^{new}_{BDSP}])$, $Cert_{BDSP}$, $Cert_{CMS}$)
{
Input: Encrypted CMS's signed message for NMS, $BDSP$'s certificate, CMS's certificate
Output: Accept or Reject
Domain Parameters: $\mathbb{H}_2(\cdot), \mathbb{H}_1(\cdot), \mathcal{G}, q$
Data: $K_{CMS,NMS}, SID_{NMS,CMS}, NMS_{ID}, \mathcal{MTU}', \lambda^1_{BDSP}, \lambda^2_{BDSP}, Cert_{NMS}, R_{NMS}$
1. Retrieve Q_{CMS} from $Cert_{CMS}$ /* CMS's public key obtained from CMS's certificate */
2. Retrieve Q_{BDSP} from $Cert_{BDSP}$ /* $BDSP$'s public key obtained from $BDSP$'s certificate */
3. Retrieve λ^{new}_{BDSP} from $E(K_{CMS,NMS} : [\lambda^{new}_{BDSP}])$ /* Decrypt $E(K_{CMS,NMS} : [\lambda^{new}_{BDSP}])$ using $NMS-CMS$'s shared secret key */
4. Compute $Temp^N_1 := \lambda^{new}_{BDSP} \cdot \mathcal{G}$
5. Compute $Temp^N_2 := \lambda^2_{BDSP} \cdot \mathcal{G} + Q_{CMS} \cdot \mathbb{H}_2(\mathbb{H}_1(SID_{NMS,CMS}))$ [e]
if($Temp^N_1 = Temp^N_2$)
{
6. Compute $SK := R_{NMS} \cdot \lambda^1_{BDSP} := R_{NMS} \cdot R_{BDSP} \cdot \mathcal{G}$
7. return Accept //accept $BDSP$ and CMS //
}
else
{
8. return Reject // reject $BDSP$ and CMS //
}
}

[e] $\lambda^2_{BDSP} \cdot \mathcal{G} = \lambda^1_{BDSP} \cdot X_1 + Q_{BDSP} \cdot X_2$, where $X_1 = \mathbb{H}_2(\mathbb{H}_1(\mathcal{MTU}'))$ and $X_2 = \mathbb{H}_2(\lambda^1_{BDSP})$

FIGURE 7. ***BDSP*** **and** ***CMS*** **verification process at** ***NMS*****.**

function ncmsVerif($E(BDSP^*_{PWD} : [\mathcal{MTU}', \mu^{new}_{NMS}]), \mu^1_{NMS}$, $Cert_{NMS}, Cert_{CMS}$)
{
Input: Encrypted CMS's signed message for $BDSP$, NMS's random nonce, NMS's certificate, CMS's certificate
Output: Accept or Reject
Domain Parameters: $\mathbb{H}_2(\cdot), \mathbb{H}_1(\cdot), \mathcal{G}, q$
Data: $BDSP^*_{PWD}, K_{CMS,BDSP}, NMS_{ID}, \mathcal{MTU}', \lambda^1_{BDSP}, \lambda^2_{BDSP}, C_{BDSP}, R_{BDSP}$
1. Retrieve Q_{CMS} from $Cert_{CMS}$ /* CMS's public key obtained from CMS's certificate */
2. Retrieve Q_{NMS} from $Cert_{NMS}$ /* NMS's public key obtained from NMS's certificate */
3. Retrieve μ^{new}_{NMS} from $E(BDSP^*_{PWD} : [\mathcal{MTU}', \mu^{new}_{NMS}])$ /* Decrypt $E(BDSP^*_{PWD} : [\mathcal{MTU}', \mu^{new}_{NMS}])$ using $BDSP-CMS$'s shared secret key */
4. Compute $Temp^B_1 := \mu^{new}_{NMS} \cdot \mathcal{G}$
5. Compute $Temp^B_2 := Y_1 \cdot \mu^1_{NMS} + Y_2 \cdot Q_{NMS} + Y_3 \cdot Q_{CMS}$ [f]
if($Temp^B_1 = Temp^B_2$)
{
6. Compute $SK := R_{BDSP} \cdot \mu^1_{NMS} := R_{BDSP} \cdot R_{NMS} \cdot \mathcal{G}$
7. return Accept //accept $BDSP$ and CMS //
}
else
{
8. return Reject // reject $BDSP$ and CMS //
}
}

[f] $Y_1 = \mathbb{H}_2(\mathbb{H}_1(NMS_{ID}))$, $Y_2 = \mathbb{H}_2(\mu^1_{NMS})$ and $Y_3 = \mathbb{H}_2(\mathbb{H}_1(K_{CMS,BDSP}))$

FIGURE 8. ***NMS*** **and** ***CMS*** **verification process at** ***BDSP*****.**

BDSP needs to verify $\mu^{new}_{NMS} \cdot \mathcal{G} = Y_1 \cdot \mu^1_{NMS} + Y_2 \cdot Q_{NMS} + Y_3 \cdot Q_{CMS}$, where $Y_1 = \mathbb{H}_2(\mathbb{H}_1(NMS_{ID}))$, $Y_2 = \mathbb{H}_2(\mu^1_{NMS})$ and $Y_3 = \mathbb{H}_2(\mathbb{H}_1(K_{CMS,BDSP}))$. To satisfy the condition, it must satisfies $Y_1 \cdot \mu^1_{NMS} = Y_1 \cdot R_{NMS} \cdot \mathcal{G}$, $Y_2 \cdot Q_{NMS} = Y_2 \cdot S_{NMS} \cdot \mathcal{G}$ and $Y_3 \cdot Q_{CMS} = Y_3 \cdot S_{CMS} \cdot \mathcal{G}$.

c: SERVICE SERVERS' REGISTRATION PHASE

After establishment of $SK_{BDSP,NMS}$ between *BA*/*BDSP* and *NMS*, *HSA* starts a new session with *NMS*. In this regards, *BDSP* can securely enrol all the service servers mainly NS_j (responsible for Big Data storage services) and JT_j (responsible for Big Data processing services) with HEAP-KDC via *NMS*. For simplicity, in this study, we assume that the *BDSP* wants to configure a Hadoop Cluster HC_j which consists of only two service servers namely (1) NS_j: responsible for controlling both namespace management and Big Data storage service activities and (2) JT_j: responsible for both task assignment and Big Data processing activities. To initiate a service server registration process, *BDSP* sends an initial enrolment request for both NS_j and JT_j to *NMS* via secure channel (using $SK_{BDSP,NMS}$). In this regard, *NMS* asks *CMS* to provide two shared symmetric keys for NS_j and JT_j. *CMS* sends the keys in encrypted format as $K_{ns,jt} = E(K_{CMS,BDSP} : [K_{CMS,NS} || K_{CMS,JT}])$. Thereafter, as a response to the *BDSP*'s request, *NMS* sends two random numbers namely $R^{NMS}_{NS_j}$ and $R^{NMS}_{JT_j}$ along with $K_{ns,jt}$ to *BDSP*. A diagram summarizing several communication message exchanges between *BDSP* and *NMS* involved throughout the service server registration process are shown in Figure 10, and it contains the following steps:

Step SSRG1: *BDSP* enters the security credentials namely the identity, symmetric key and synthetic password for both the service servers (i.e., NS_j and JT_j) into *HSA*. *HSA* computes the masked identities for NS_j and JT_j as $TNS_{ID} = h(NS_{ID} || HSA_{ID} || rss_1)$ and $TJT_{ID} = h(JT_{ID} || HSA_{ID} || rss_2)$, and the masked password for the same service servers as $SID^j_{NS} = h(NS_{PWD} || rss_1 || R^{NMS}_{NS_j})$ and $SID^j_{JT} = h(JT_{PWD} || rss_2 || R^{NMS}_{JT_j})$, respectively. Note that rss_1 and rss_2 are two random numbers chosen by *HSA*.

Step SSRG2: *HSA* computes the masked identity of BDSP as $\mathcal{MTU}'$ and construct a message $M_5 = \{\mathcal{MTU}', NMS_{ID}, n_3, E(K_{(CMS,BDSP)} : [\mathcal{MTU}', n_3, TNS_{ID}, K_{CMS,NS}, SID^j_{NS}, TJT_{ID}, K_{CMS,JT}, SID^j_{JT}]), E(SK_{BDSP,NMS} : [\mathcal{MTU}', n_3])\}$ and sends the same to *NMS*.

Step SSRG3: After receiving the message M_5, *NMS* checks the presence of $\mathcal{MTU}'$ into $Z_5 = E(SK_{BDSP,NMS} : [\mathcal{MTU}', n_3])$ after decrypting the Z_5 using the key $SK_{BDSP,NMS}$. If $\mathcal{MTU}'$ exists in its database then *BDSP* is treated as authentic service provider and *NMS* broadcasts $\{\mathcal{MTU}', E(K_{(CMS,BDSP)} : [\mathcal{MTU}', n_3, TNS_{ID}, K_{CMS,NS}, SID^j_{NS}, TJT_{ID}, K_{CMS,JT}, SID^j_{JT}])\}$ to both *CMS* and *ES*. After receiving $E(K_{(CMS,BDSP)} : [\mathcal{MTU}', n_3, TNS_{ID}, K_{CMS,NS}, SID^j_{NS}, TJT_{ID}, K_{CMS,JT}, SID^j_{JT}])\}$, both *CMS* and *ES* decrypts it and updates their service server databases (after finding out the availability of masked server identity into their databases) and sends their acknowledgements to *NMS*.

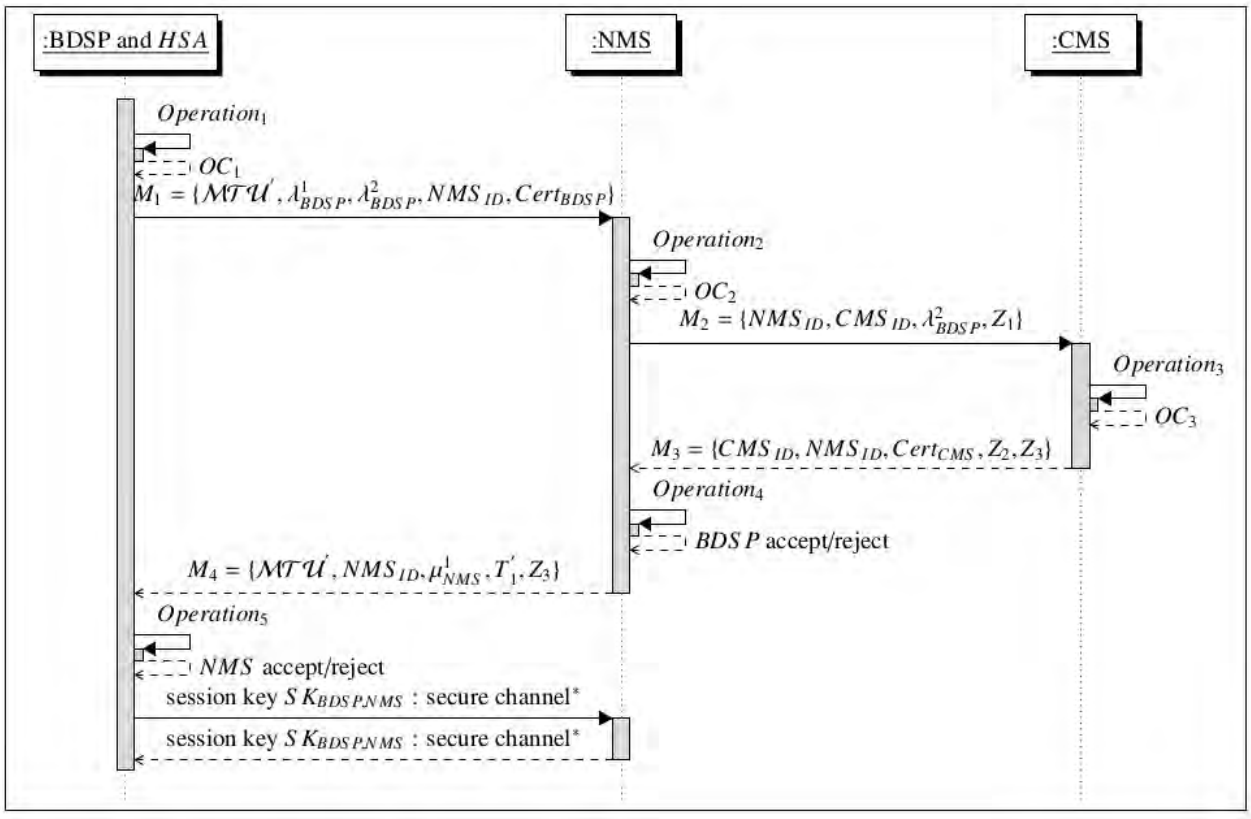

FIGURE 9. Summary of authenticated key agreement process between *BDSP*/*BA* and *NMS*. Note: Here, $T_1' = \{Cert_{CMS}, Cert_{NMS}\}$, $Z_1 = E(K_{CMS,NMS} : [\mathcal{MTU}', \mu^2_{NMS}])$, $Z_2 = E(K_{CMS,NMS} : [\lambda^{new}_{BDSP}])$, $Z_3 = E(BDSP^*_{PWD} : [\mathcal{MTU}', \mu^{new}_{NMS}])$, $Operation_1$, $Operation_2$, $Operation_3$, $Operation_4$ and $Operation_5$: signifies execution of the function bdspSignPairGen[ ](·), nmsSignPairGen[ ](·), cmodifiedSignPairGen[ ](·), bcmsVerif(·) and ncmsVerif(·), respectively (refer Figure 4, Figure 5, Figure 6, Figure 7 and Figure 8) and OC_1, OC_2 and OC_3: denotes outcome of bdspSignPairGen[ ](·), nmsSignPairGen[ ](·) and cmodifiedSignPairGen[ ](·) functions respectively.

Step SSRG4: Upon getting the acknowledgements from both the servers (*CMS* and *ES*), *NMS* checks $n_3' \stackrel{?}{=} n_3$. If the condition is satisfied then *NMS* constructs a service servers' (here, we consider two service servers i.e., *NS* and *JT* in a particular cluster) registration completion message as $M_6 = \{NMS_{ID}, \mathcal{MTU}', n_3, E(SK_{BDSP,NMS} : [NMS_{ID}, n_3])\}$ and sends it to *BDSP*.

Step SSRG5: Getting the message M_6, *BDSP* decrypts $E(SK_{BDSP,NMS} : [NMS_{ID}, n_3])$ and checks that $n_3' \stackrel{?}{=} n_3$. If the condition is satisfied then *BDSP* understand the legitimacy of *NMS* and realized that the service servers registration has been successfully accomplished with HEAP-KDC.

Step SSRG6: *HSA* computes $NS'_{ID} = NS_{ID} \oplus h(BDSP_{ID} || BDSP_{PWD})$, $JT'_{ID} = JT_{ID} \oplus h(BDSP_{ID} || BDSP_{PWD})$, $NS_j^{PWD'} = NS_j^{PWD} \oplus h(BDSP_{ID} || BDSP_{PWD})$, $JT_j^{PWD'} = JT_j^{PWD} \oplus h(BDSP_{ID} || BDSP_{PWD})$, $rss_1^* = rss_1 \oplus h(BDSP_{ID} || BDSP_{PWD})$ and $rss_2^* = rss_2 \oplus h(BDSP_{ID} || BDSP_{PWD})$, respectively. *HSA* stores NS'_{ID}, JT'_{ID}, $NS_j^{PWD'}$, $JT_j^{PWD'}$, rss_1^*, rss_2^* and HC_j^{ID} information into its database for future use. Note that HC_j^{ID} signifies here as the pre-deployed Hadoop cluster's identity and *HSA* assigns a random value for it.

After successful registration of the service servers, *BDSP* configures the service servers and its client application ($HDCL_j$) for the cluster HC_j. In this regard, *BDSP* stores the secret credentials offline among the corresponding service servers and $HDCL_j$. More precisely NS_j has $TNS_{ID} = h(NS_{ID} || HSA_{ID} || rss_1)$, $K_{CMS,NS}$ and $SID^j_{NS} = h(NS_{PWD} || rss_1 || R^{NMS}_{NS_j})$; JT_j has $TJT_{ID} = h(JT_{ID} || HSA_{ID} || rss_2)$, $K_{CMS,JT}$ and $SID^j_{JT} = h(JT_{PWD} || rss_2 || R^{NMS}_{JT_j})$, and $HDCL_j$ has both $TNS_{ID} = h(NS_{ID} || HSA_{ID} || rss_1)$ and $TJT_{ID} = h(JT_{ID} || HSA_{ID} || rss_2)$ information, respectively. Thus, completes the HC_j's deployment process. Finally, *BDSP* make the HC_j online for providing the Big Data storage and processing services to the end users'.

Remark 1: In the service server registration phase, we present an enrolment strategy considering two service servers (Namenode Server (*NS*) and Job Tracker (*JT*)) belong to a particular Hadoop cluster (HC_j). But, for simplicity and better understanding, throughout this paper, we consider only one service server to describe the other phases of the proposed

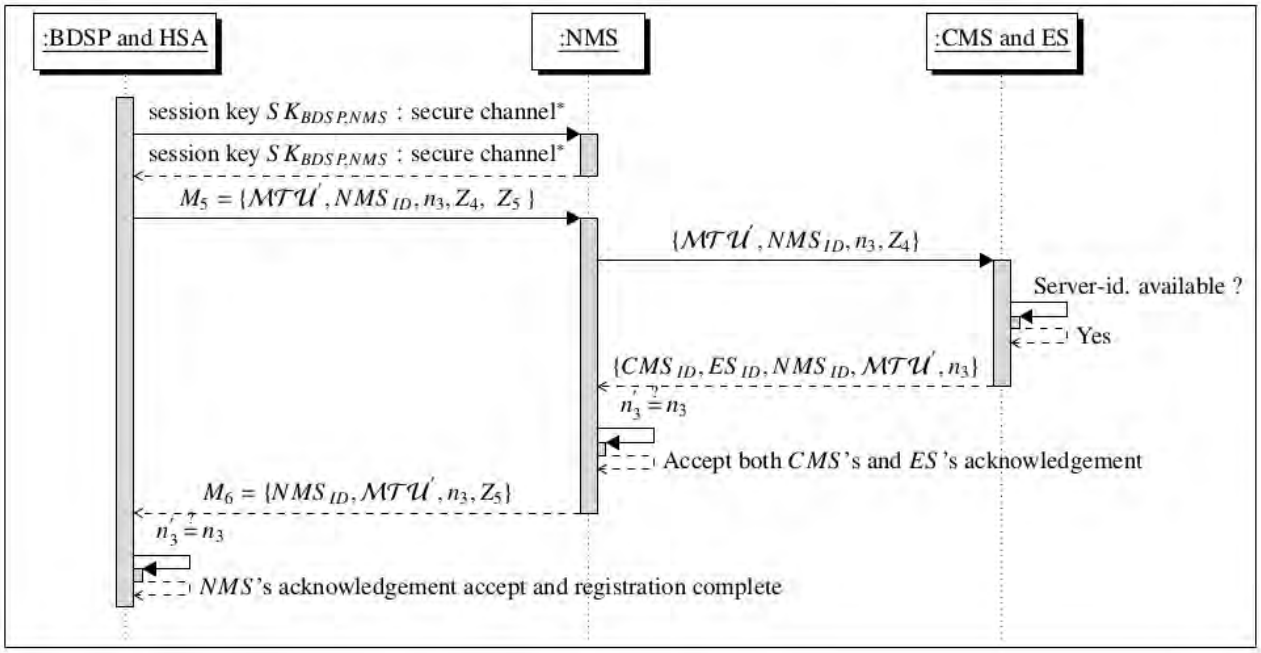

FIGURE 10. Summary of service server registration process. **Note:** Here, $Z_4 = E(K_{(CMS,BDSP)} : [\mathcal{MTU}', n_3, TNS_{ID}, SID^j_{NS}, TJT_{ID}, SID^j_{JT}])$ and $Z_5 = E(SK_{BDSP,NMS} : [NMS_{ID}, \mathcal{MTU}', n_3])$.

protocol. Note that *NS* is responsible for providing Big Data storage services utilizing HDFS and *JT* is solely made for providing Big Data processing services using MapReduce to the end users'. Though, we restrict our discussion with *NS* only, but one could simply apply the same proposed mechanism for *JT* also.

Remark 2: According to the policy of the proposed scheme, the online enrolment process of the service servers' (*NS*s' or *JT*s' belongs to a particular cluster (HC_j)) is simple, flexible and scalable. Because, a service provider ($BDSP_j$) can add or remove any number of service servers' (*NS*s' or *JT*s') to or form HEAP-KDC. However, to do this, the service provider needs to enroll himself with the HEAP-KDC once and keep his user identity and password secret.

4) USER REGISTRATION

Initially, a user C_i needs to register himself with *ES* offline (i.e., via the out-of-band channel or postal network) by giving his identity proof, address proof and service subscription payment documents. This entails C_i to avail a dummy user-id ($D_{C_{ID}}$), dummy password (D_{PWD}) and a one-time key ($K_{(NMS,C)}$) via the out-of-band channel to his physical address. Before sending these three security credentials to C_i, *ES* securely sends $D_{C_{ID}}$ and D_{PWD} to *CMS* for creating a dummy account of C_i. *ES* also sends securely the one-time key $K_{(NMS,C)}$ to *NMS*. Note that C_i needs to use $D_{C_{ID}}$, D_{PWD} and $K_{(NMS,C)}$ only for once to create his own user profile into *NMS* with the help of *CMS*. But, C_i is permissible to use the key $K_{(NMS,C)}$ (we also call it as pass-phrase) at the time of session key formation and password updation tasks. Figure 11 summarizes the user registration phase and contains the following steps:

Step URG1: C_i enters $D_{C_{ID}}$ and D_{PWD} into HCA_j. HCA_j generates a random nonce say n_1. HCA_j encrypts $D_{C_{ID}}$ and n_1 using D_{PWD} as $\mathcal{U}_{Dtl} = E(D_{PWD} : [D_{C_{ID}}, n_1])$. HCA_j sends the message $msg_{C1} = \{CMS_{ID}, n_1, \mathcal{U}_{Dtl}\}$ to the *CMS*.

Step URG2: After receiving msg_{C1}, *CMS* decrypts $\mathcal{U}_{Dtl}$ and checks the availability of $D_{C_{ID}}$ in its database. If it exists then *CMS* acknowledge with a message $msg_{C2} = \{NMS_{ID}, CMS_{ID}, n_1\}$ to C_i and C_i is permissible to create his own profile into *NMS*, and goto Step URG3 else, reject C_i's request.

Step URG3: C_i enters a new user-id C_{ID} in HCA_j. HCA_j chooses a random secret r_i and transforms the C_{ID} as $\mathcal{TU} = h(C_{ID}||r_i||HCA_{ID})$. Using this transformed identity $\mathcal{TU}$, both C_i and *CMS* establishes a shared secret key say $SK_{(CMS,C)}$ between themselves using the similar analogy proposed as "Initial key establishment between C_i and *FEAS*" in DPTSAP [10]. It is worth noting that HCA_{ID} is the public identity of client application and it is known to HCA_j.

Step URG4: C_i enters his password *PWD* into HCA_j. HCA_j computes the masked password $PWD^* = h(h(r_i||PWD)||R^{CMS}_{C_i})$ and go to Step URG5. Here, $R^{CMS}_{C_i}$ was generated by *CMS* and it has been delivered securely using the key $SK_{(CMS,C)}$ at the time of "secret key establishment" process [10]. Further, both C_i's request and *CMS*'s response messages using DPTSAP [10] are represented as msg_{C3} and msg_{C4} respectively in Figure 11.

Step URG5: HCA_j asks C_i to give his registered mobile number ($MBNO_C$) and a valid email id. (EID_C). After taking $MBNO_C$ and EID_C from C_i, HCA_j encrypts $\mathcal{MTU}$, PWD^* $MBNO_C$ and EID_C using $K_{NMS,C}$ as

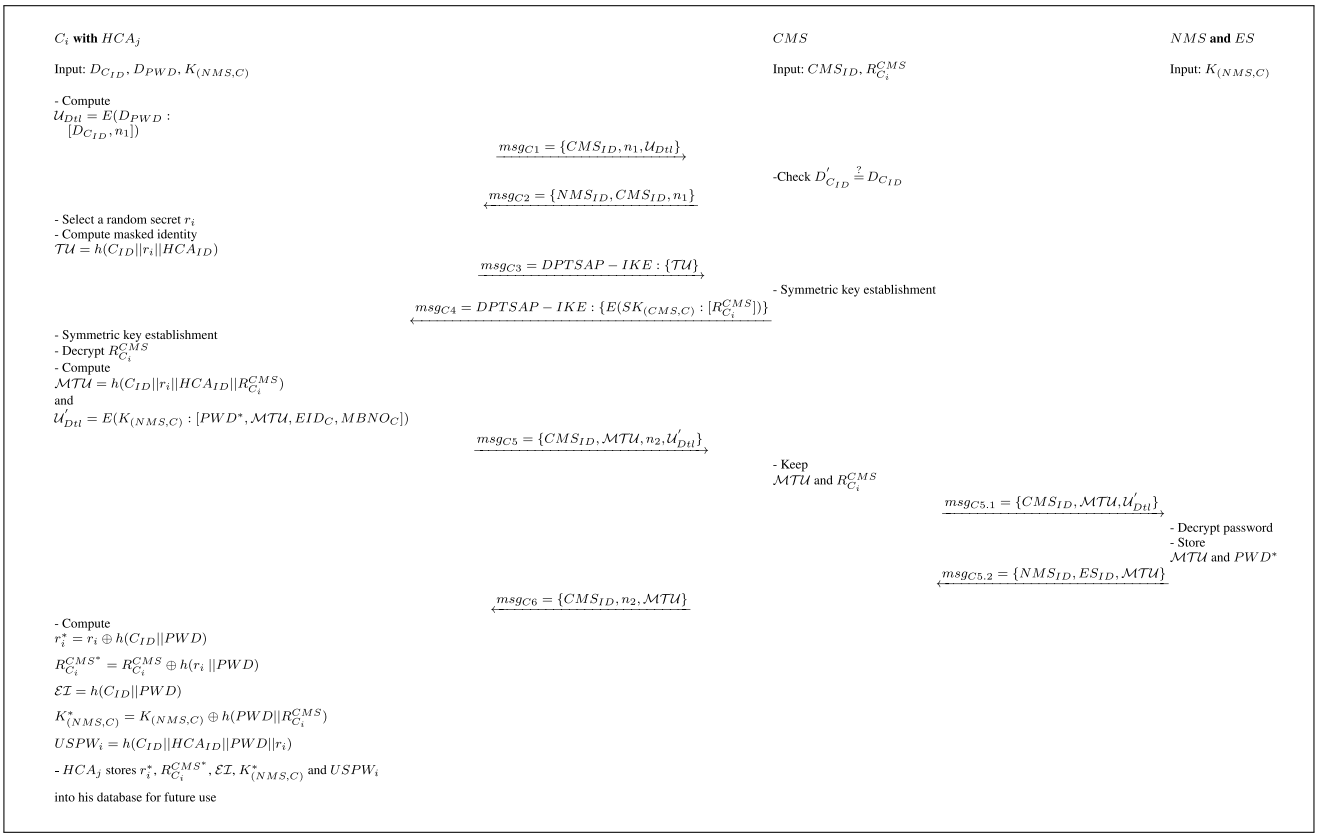

FIGURE 11. Summary of user registration.

$\mathcal{U}'_{Dtl} = E(K_{(NMS,C)} : [PWD^*, \mathcal{MTU}, MBNO_C])$, where $\mathcal{MTU} = h(C_{ID}||r_i\ ||HCA_{ID}||\ R_{C_i}^{CMS})$. HCA_j sends the message $msg_{C5} = \{CMS_{ID}, \mathcal{MTU}, n_2, \mathcal{U}'_{Dtl}\}$ to the CMS.

Step URG6: After receiving msg_{C5}, CMS broadcasts both $\mathcal{MTU}$ and $\mathcal{U}'_{Dtl}$ to both ES and NMS servers respectively. CMS keeps only $\mathcal{MTU}$ and $R_{C_i}^{CMS}$ in its database and deletes other information, wherein, after decrypting $\mathcal{U}'_{Dtl}$, both ES and NMS stores $\mathcal{MTU}$, EID_C, $MBNO_C$ and PWD^* into their corresponding databases. Finally, CMS sends a registration confirmation message to C_i through HCA_j and goto Step URG7.

Step URG7: HCA_j computes $r_i^* = r_i \oplus h(C_{ID}\ ||PWD)$, $R_{C_i}^{CMS^*} = R_{C_i}^{CMS} \oplus h(r_i\ ||PWD)$, $\mathcal{EI} = h(C_{ID}||\ PWD)$, $K^*_{(NMS,C)} = K_{(NMS,C)} \oplus h(PWD\ ||\ R_{C_i}^{CMS})$ and $USPW_i = h(C_{ID}\ ||HCA_{ID}\ ||PWD\ ||r_i)$, and then stores these information into HCA_j's database. HCA_j will use these information at the time of user authentication and password updation phases.

After successful registration of client C_i with HEAP-KDC, C_i needs to remember only two parameters C_{ID} and PWD to access the services from any on-demand service server (either NS_j or JT_j) belongs to a particular Hadoop cluster say HC_j followed by a user login and authenticated key agreement phases discussed in the following sections. Thus, the proposed protocol is user friendly in nature.

5) USER LOGIN AT WORKSTATION

After the completion of user registration process, C_i tries to login into the system using the client application HCA_j. Before initiating user login task into C_i's workstation, HCA_j loads the user's masked identities ($\mathcal{EI}_i$, where $i = 1, 2, \cdots, n$) into browser cookies. The client-side login process contains the following steps:

Step UL1: C_i enter his original identity C_{ID} and password PWD into HCA_j. HCA_j computes $\mathcal{EI}^* = h(C_{ID}||PWD)$ and checks this entry exists in the cookies or not. If it exists then HCA_j loads $r_i^*, R_{C_i}^{CMS^*}$ and $USPW_i$ entries for the same C_i. HCA_j computes $r_i = r_i^* \oplus h(C_{ID}\ ||PWD)$ and $R_{C_i}^{CMS} = R_{C_i}^{CMS^*} \oplus h(r_i\ ||PWD)$ and goto Step UL2. Else, client can repeat Step UL1 with other user identity and password.

Step UL2: HCA_j computes $USPW_i^* = h(C_{ID}\ ||\ HCA_{ID}\ ||\ PWD\ ||\ r_i)$ and checks if the condition $USPW_i^* = USPW_i$ holds or not. If it holds, C_i is treated as authentic user.

After logged in into the client system, C_i needs to select the j^{th} cluster say HC_j from a list of Hadoop clusters $(HC_1, HC_2, \cdots, HC_j, \cdots, HC_n)$ for storing or processing the Big Data. According to the selection, HCA_j loads the respective client application say $HDCL_j$ belongs to the HC_j into the client workstation. In order to store or process C_i's Big Data securely, HCA_j initiate a single sign-on and

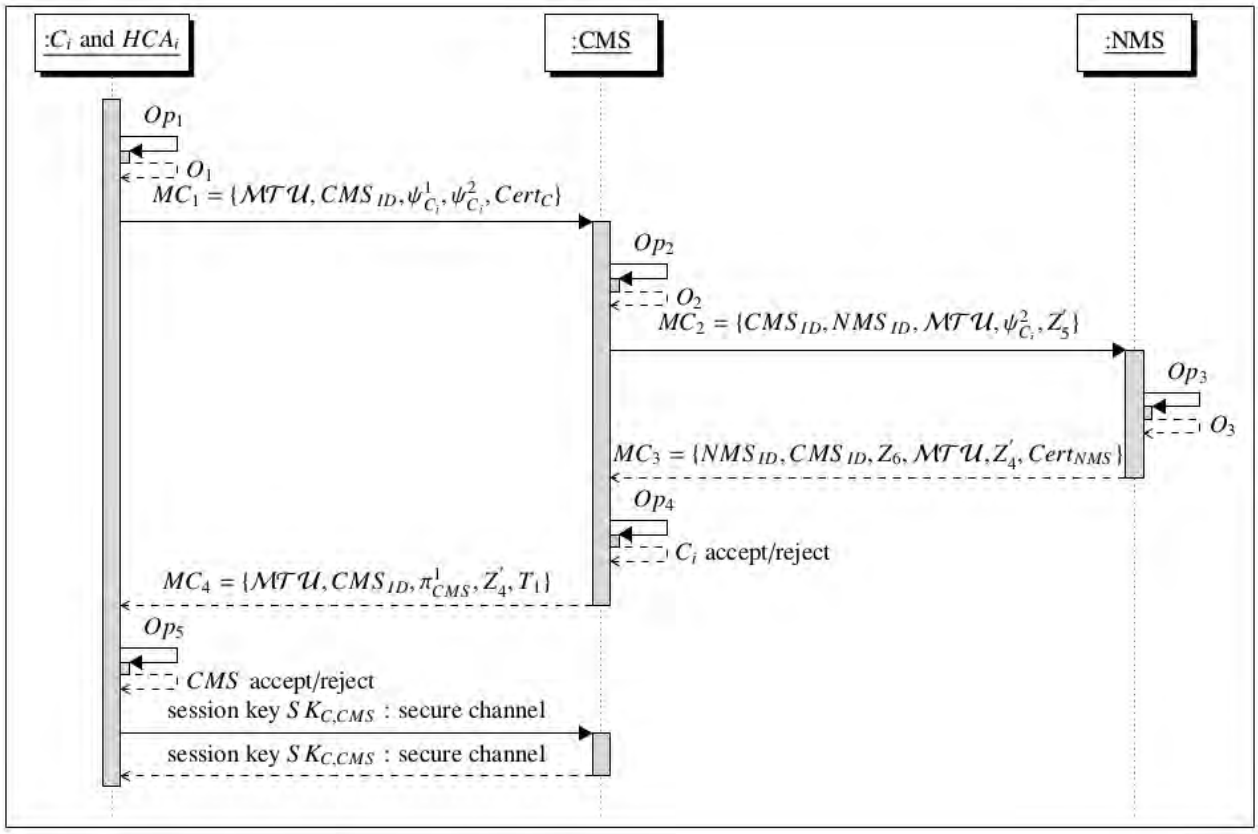

FIGURE 12. Summary of mutual authentication and key establishment process between C_i and CMS. Note: $T_1 = \{Cert_{CMS}, Cert_{NMS}\}$, $Z'_4 = E(PWD^* : [\mathcal{MTU}||V_{cnew}])$, $Z'_5 = E(K_{CMS,NMS} : [CMS_{ID}||\pi^2_{CMS}])$, $Z_6 = E(K_{CMS,NMS} : [CMS_{ID}||V_{cmsnew}])$, Op_i, where $i = \{1, 2, 3, 4, 5\}$: signifies the execution of the function *cSignPairGen[ ](·), cmsSignPairGen[ ](·), nmodifiedSignPairGen[ ](·), cnmsVerif(·)* and *cmnmVerif(·)*, respectively (refer Figure 13, Figure 14, Figure 15, Figure 16 and Figure 17) and O_j, where $j = \{1, 2, 3\}$: denotes the outcome of *cSignPairGen[ ](·), cmsSignPairGen[ ](·)* and *nmodifiedSignPairGen[ ](·)* functions.

session key establishment task between C_i and HEAP-KDC as follows.

6) SINGLE SIGN-ON AND DYNAMIC KEY ESTABLISHMENT

A two-server based Single sign-on (SSO) and Dynamic Key Agreement (DKA) process is proposed to access Big Data storage services from a remote Namenode Server (NS_j) (or access Big Data processing services from a remote Job Tracker (JT_j)) of a specified Hadoop cluster say HC_j. The proposed two-server based SSO and dynamic key agreement process establishes a dynamic key between C_i and HEAP-KDC followed by a mutual authentication process. The detail steps involved in this tasks are discussed as follows.

Note here, the proposed SSO and dynamic key establishment between C_i and CMS are based on ECC and AES. The known pre-deployed security domain parameters among HCA_i, CMS and NMS are $\mathcal{G}$, q, $\mathbb{H}_1(\cdot)$ and $\mathbb{H}_2(\cdot)$, respectively. Initially, HCA_i selects a private key S_C for client C_i and computes the corresponding public key $Q_C = S_C \cdot \mathcal{G}$. Similarly, S_{CMS} and S_{NMS} are private keys for CMS and NMS and the corresponding public keys are $Q_{CMS} = S_{CMS} \cdot \mathcal{G}$ and $Q_{NMS} = S_{NMS} \cdot \mathcal{G}$, respectively. Here, S_C, S_{CMS} and S_{NMS} are belongs to $\mathbb{Z}_q{}^*$. All these parties keep their private keys (i.e., S_C, S_{CMS} and S_{NMS}) secret, but disseminate the public keys Q_C, Q_{CMS} and Q_{NMS} using $Cert_E$. The proposed key agreement and mutual authentication task between C_i and CMS are summarized in Figure 12.

The detail steps involved in this process are discussed as follows.

Step DKA1: HCA_j computes C_i masked identity $\mathcal{MTU} = h(C_{ID}||r_i \ ||HCA_{ID}||\ R^{CMS}_{C_i})$. Further, HCA_j generates two pseudo-random numbers ($\psi^1_{C_i}$ and $\psi^2_{C_i}$) utilizing $\mathcal{MTU}$ and other pre-loaded security domain parameters as the input to a function say *cSignPairGen[ ](·)* (see Figure 13).

Step DKA2: HCA_j construct a message $MC_1 = \{\mathcal{MTU}, CMS_{ID}, \psi^1_{C_i}, \psi^2_{C_i}, Cert_C\}$ and sends it to CMS via a public channel.

Step DKA3: After receiving MC_1, CMS searches its database to check the existence of $\mathcal{MTU}$. If it finds the same then CMS generates two pseudo-random numbers (π^1_{CMS} and π^2_{CMS}) by taking CMS_{ID} and other domain parameters as the input to a function say *cmsSignPairGen[ ](·)* (see Figure 14).

Step DKA4: CMS constructs a message $MC_2 = \{CMS_{ID}, NMS_{ID}, \mathcal{MTU}, \psi^2_{C_i}, E(K_{CMS,NMS} : [CMS_{ID}||\ \pi^2_{CMS}])\}$ and sends it to NMS via a public channel.

function *cSignPairGen[ ]*$(\mathcal{MTU}, S_C)$
{
Input: Client's masked identity and Client's private key
Output: Two ECC points as $\psi^1_{C_i}$ and $\psi^2_{C_i}$
Domain Parameters: $\mathbb{H}_2(\cdot), \mathbb{H}_1(\cdot), \mathcal{G}, q$
Data: $PWD^*, K_{NMS,C}, CMS_{ID}, \mathcal{MTU}$
1. Selects a pseudo-random number $R_C \in \mathbb{Z}^*_q$
2. Compute $\psi^1_{C_i} := R_C \cdot \mathcal{G}$
3. Compute $\psi^2_{C_i}$ [g] $:= R_C \cdot \mathbb{H}_2(\mathbb{H}_1(\mathcal{MTU})) + S_C \cdot \mathbb{H}_2(\psi^1_{C_i}) \pmod q$
4. return $R_C, \psi^1_{C_i}, \psi^2_{C_i}$
}

[g]Note: C_i digitally sign on both $\mathcal{MTU}$ and $\psi^1_{C_i}$ using its private key S_C.

FIGURE 13. C_i**'s signature generation process.**

function *cmsSignPairGen[ ]*(CMS_{ID}, S_{CMS})
{
Input: CMS's identity and CMS's private key
Output: Two ECC points π^1_{CMS} and π^2_{CMS}
Domain Parameters: $\mathbb{H}_2(\cdot), \mathbb{H}_1(\cdot), \mathcal{G}, q$
Data: $K_{CMS,NMS}, SID_{NMS,CMS}, NMS_{ID}, CMS_{ID}, \mathcal{MTU}$
1. Choose a pseudo-random number $R_{CMS} \in \mathbb{Z}^*_q$
2. Compute $\pi^1_{CMS} := R_{CMS} \cdot \mathcal{G}$
3. Compute π^2_{CMS} [h] $:= R_{CMS} \cdot \mathbb{H}_2(\mathbb{H}_1(CMS_{ID})) + S_{CMS} \cdot \mathbb{H}_2(\pi^1_{CMS}) \pmod q$
4. return $R_{CMS}, \pi^1_{CMS}, \pi^2_{CMS}$
}

[h]Note: CMS digitally sign on both CMS_{ID} and π^1_{CMS} using its private key S_{CMS}.

FIGURE 14. ***CMS*****'s signature generation process.**

Step DKA5: After receiving the message MC_2, *NMS* searches both $\mathcal{MTU}$ and CMS_{ID} in its database. If both are exists then *NMS* understands that both C_i and *CMS* are legitimate parties, and goto Step DKA6; otherwise, rejects *CMS*'s request.

Step DKA6: *NMS* loads both C_i's pass-phrase ($K_{NMS,C}$) and *NMS*'s secret identity ($SID_{CMS,NMS}$) from its database. *NMS* modifies both C_i's and *CMS*'s partial signatures (i.e., π^2_{CMS} and $\psi^2_{C_i}$) using a function *nmodifiedSignPairGen[ ]*$(\cdot)$ (see Figure 15) as (1) V_{cnew} [3] $= \pi^2_{CMS} + S_{NMS} \cdot \mathbb{H}_2(\mathbb{H}_1(K_{NMS,C})) \pmod q = R_{CMS} \cdot \mathbb{H}_2(\mathbb{H}_1(CMS_{ID})) + S_{CMS} \cdot \mathbb{H}_2(\pi^1_{CMS}) + S_{NMS} \cdot \mathbb{H}_2(\mathbb{H}_1(K_{NMS,C})) \pmod q$ and (2) V_{cmsnew} [4] $= \psi^2_{C_i} + S_{NMS} \cdot \mathbb{H}_2(\mathbb{H}_1(SID_{CMS,NMS})) \pmod q = R_C \cdot \mathbb{H}_2(\mathbb{H}_1(\mathcal{MTU}) + S_C \cdot \mathbb{H}_2(\psi^1_{C_i}) + S_{NMS} \cdot \mathbb{H}_2(\mathbb{H}_1(SID_{CMS,NMS})) \pmod q$ and goto Step DKA7.

Step DKA7: *NMS* constructs a message $MC_3 = \{NMS_{ID}, CMS_{ID}, E(K_{CMS,NMS} : [CMS_{ID} || V_{cmsnew}]), \mathcal{MTU}, E(PWD^* : [\mathcal{MTU} || V_{cnew}]), Cert_{NMS}\}$ and sends the same to *CMS*.

Step DKA8: *CMS* verifies the legitimacy of both C_i and *NMS* utilizing the *cnmsVerif*$(\cdot)$ function shown in Figure 16. If the function *cnmsVerif*$(\cdot)$ returns "Accept", then *CMS* construct a session key $SK_{C,CMS} = R_{CMS} \cdot \psi^1_{C_i} = R_{CMS} \cdot R_C \cdot \mathcal{G}$ and a message $MC_4 = \{\mathcal{MTU}, CMS_{ID}, \pi^1_{CMS}, E(PWD^* : [\mathcal{MTU} || V_{cnew}]), Cert_{NMS}, Cert_{CMS}\}$. *CMS* sends MC_4 to C_i, and goto Step DKA9; otherwise, it rejects C_i's request.

Step DKA9: After getting the message MC_4, C_i verifies both *NMS* and *CMS* utilizing the function *cmnmVerif*$(\cdot)$ (see Figure 17). If the function *cmnmVerif*$(\cdot)$ returns "Accept" then C_i constructs a session key $SK_{C,CMS} = R_C \cdot \pi^1_{CMS} = R_{C_i} \cdot R_{CMS} \cdot \mathcal{G}$ otherwise; rejects *NMS*'s response.

[3]Note: *NMS* digitally sign on the messages say π^2_{CMS}, $K_{NMS,C}$ and π^1_{CMS} using NMS's private key S_{NMS}.

[4]Note: *NMS* digitally sign on the messages say $\mathcal{MTU}$, $SID_{CMS,NMS}$ and $\psi^1_{C_i}$ using its private key S_{NMS}.

function *nmodifiedSignPairGen[ ]*$(\psi^2_{C_i}, E(K_{CMS,NMS} : [CMS_{ID}, \pi^2_{CMS}]))$
{
Input: C_i's partial signature and encrypted NMS's partial signature
Output: Both C_i's and NMS's modified signature
Domain Parameters: $\mathbb{H}_2(\cdot), \mathbb{H}_1(\cdot), \mathcal{G}, q$
Data: $PWD^*, K_{NMS,C}, K_{CMS,NMS}, SID_{NMS,CMS}, NMS_{ID}, CMS_{ID}, \mathcal{MTU}$
if($\mathcal{MTU}^* = \mathcal{MTU}$ && $CMS'_{ID} = CMS_{ID}$)
{
1. $V_{cnew} = \pi^2_{CMS} + S_{NMS} \cdot \mathbb{H}_2(\mathbb{H}_1(K_{NMS,C})) \pmod q$
2. $V_{cmsnew} = \psi^2_{C_i} + S_{NMS} \cdot \mathbb{H}_2(\mathbb{H}_1(SID_{CMS,NMS})) \pmod q$
3. return V_{cnew}, V_{cmsnew} // accept C_i and CMS //
}
else
{
4. return null // reject C_i and CMS //
}
}

FIGURE 15. **Signature updation process at** ***NMS*****.**

function *cnmsVerif*$(E(K_{CMS,NMS} : [CMS_{ID}, V_{cmsnew}]), Cert_C, Cert_{NMS})$
{
Input: Modified and encrypted C_i's partial signature, C_i's certificate, NMS's certificate
Output: Accept or Reject
Domain Parameters: $\mathbb{H}_2(\cdot), \mathbb{H}_1(\cdot), \mathcal{G}, q$
Data: $K_{CMS,NMS}, SID_{NMS,CMS}, NMS_{ID}, \mathcal{MTU}, \psi^1_{C_i}, \psi^2_{C_i}, Cert_{CMS}, R_{CMS}$
1. Retrieve $Q_{NMS} = S_{NMS} \cdot \mathcal{G}$ from $Cert_{NMS}$
/* NMS's public key obtained from NMS's certificate */
2. Retrieve Q_C from $Cert_C$
/* C_i's public key obtained from C_i's certificate */
3. Retrieve V_{cmsnew} from $E(K_{CMS,NMS} : [V_{cmsnew}])$
/* Decrypt $E(K_{CMS,NMS} : [V_{cmsnew}])$ using $NMS - CMS$'s shared secret key */
4. Compute $Temp_1^{CMS} := V_{cmsnew} \cdot \mathcal{G}$
5. Compute $Temp_2^{CMS} := \psi^2_{C_i} \cdot \mathcal{G} + Q_{NMS} \cdot \mathbb{H}_2(\mathbb{H}_1(SID_{NMS,CMS}))$ [i]
if($Temp_1^{CMS} = Temp_2^{CMS}$)
{
6. Compute $SK := R_{CMS} \cdot \psi^1_{C_i} := R_{CMS} \cdot R_C \cdot \mathcal{G}$
7. return Accept //accept C_i and NMS //
}
else
{
8. return Reject // reject C_i and NMS //
}
}

[i]$\psi^2_{C_i} \cdot \mathcal{G} = \psi^1_{C_i} \cdot X_{11} + Q_C \cdot X_{22}$, where $X_{11} = \mathbb{H}_2(\mathbb{H}_1(\mathcal{MTU}))$, $X_{22} = \mathbb{H}_2(\psi^1_{C_i})$ and $Q_C = S_C \cdot \mathcal{G}$.

FIGURE 16. C_i **and** ***NMS*** **verification process at** ***CMS*****.**

function *cmnmVerif*($E(PWD^*$: $[\mathcal{MTU}$, $V_{cnew}])$, π^1_{CMS}, $Cert_{NMS}, Cert_{CMS})$
{
Input: Modified and encrypted CMS's partial signature, CMS's random nonce, NMS's certificate, CMS's certificate
Output: Accept or Reject
Domain Parameters: $\mathbb{H}_2(\cdot), \mathbb{H}_1(\cdot), \mathcal{G}, q$
Data: $PWD^*, K_{C,CMS}, NMS_{ID}, \mathcal{MTU}, \psi^1_{C_i}$, $\psi^2_{C_i}, Cert_C, R_C$
1. Retrieve Q_{CMS} from $Cert_{CMS}$
/* CMS's public key obtained from CMS's certificate */
2. Retrieve Q_{NMS} from $Cert_{NMS}$
/* NMS's public key obtained from NMS's certificate */
3. Retrieve V_{cnew} from $E(PWD^* : [\mathcal{MTU}, V_{cnew}])$
/* Decrypt $E(PWD^* : [\mathcal{MTU}, V_{cnew}])$ using $C_i - CMS$'s shared secret key */
4. Compute $Temp^C_1 := V_{cnew} \cdot \mathcal{G}$
5. Compute $Temp^C_2 := Y_{11} \cdot \pi^1_{CMS} + Y_{22} \cdot Q_{CMS} + Y_{33} \cdot Q_{NMS}$
if$(Temp^C_1 = Temp^C_2)$
{
6. Compute $SK := R_C \cdot \pi^1_{CMS} := R_C \cdot R_{CMS} \cdot \mathcal{G}$
7. return Accept //accept CMS and NMS //
}
else
{
8. return Reject // reject CMS and NMS //
}
}

$^jY_{11} = \mathbb{H}_2(\mathbb{H}_1(CMS_{ID}))$, $Y_{22} = \mathbb{H}_2(\pi^1_{CMS})$ and $Y_{33} = \mathbb{H}_2(\mathbb{H}_1(K_{C,CMS}))$.

FIGURE 17. ***NMS*** **and** ***CMS*** **verification process in C_i's workstation.**

After establishment of the dynamic key $SK_{C,CMS}$ between C_i and *CMS*, HCA_j allows C_i to move further for requesting service server ticket as follows.

Proof of Correctness: In order to verify the legitimacy of both C_i and *NMS*, *CMS* needs to check $V_{cmsnew} \cdot \mathcal{G} = \psi^2_{C_i} \cdot \mathcal{G} + Q_{NMS} \cdot \mathbb{H}_2(\mathbb{H}_1(SID_{NMS,CMS}))$. To satisfy the verification condition, it must holds $\psi^2_{C_i} \cdot \mathcal{G} = \psi^1_{C_i} \cdot \mathbb{H}_2(\mathbb{H}_1(\mathcal{MTU})) + Q_C \cdot \mathbb{H}_2(\psi^1_{C_i}) = R_C \cdot \mathbb{H}_2(\mathbb{H}_1(\mathcal{MTU})) \cdot \mathcal{G} + S_C \cdot \mathbb{H}_2(\psi^1_{C_i}) \cdot \mathcal{G}$ and $Q_{NMS} \cdot \mathbb{H}_2(\mathbb{H}_1(SID_{NMS,CMS})) = S_{NMS} \cdot \mathbb{H}_2(\mathbb{H}_1(SID_{NMS,CMS})) \cdot \mathcal{G}$. Similarly, to verify the legitimacy of both *NMS* and *CMS*, C_i needs to verify $V_{cnew} \cdot \mathcal{G} = Y_{11} \cdot \pi^1_{CMS} + Y_{22} \cdot Q_{CMS} + Y_{33} \cdot Q_{NMS}$. To satisfy the condition, it must satisfies $Y_{11} \cdot \pi^1_{CMS} = Y_{11} \cdot R_{CMS} \cdot \mathcal{G}$, $Y_{22} \cdot Q_{CMS} = Y_{22} \cdot S_{CMS} \cdot \mathcal{G}$ and $Y_{33} \cdot Q_{NMS} = Y_{33} \cdot S_{NMS} \cdot \mathcal{G}$.

7) BIG DATA STORAGE SERVICE SERVER TICKET GRANTING

In this process, C_i requests for a Big Data storage service ticket from *CMS*. Before doing so, HCA_j provides a drop-down list from which C_i needs to select a particular Hadoop cluster say HC_j. After selecting the cluster, HCA_j automatically choose a Namenode Server (NS_j) for client C_i. Finally, HCA_j initiates the service server ticket granting process. A diagram summarizing the message exchanges between C_i and *CMS* involved throughout the service ticket granting process is shown in Figure 18, and it contains the following steps:

Step SSTG1: HCA_j constructs a message $MC_5 = \{\mathcal{MTU}, TNS_{ID}, E(SK_{C,CMS} : [\mathcal{MTU}, n_4, \psi^2_{C_i}])\}$ into C_i's workstation. HCA_j sends MC_5 to *CMS*.

Step SSTG2: After getting MC_5, *CMS* checks the presence of both $\mathcal{MTU}$ and $\psi^2_{C_i}$ in its system cache. If both exists then *CMS* constructs two tokens (namely Client Token (CT) and Namenode Server Token (NST)) namely $CT = C_{token} = E(SK_{(C,CMS)} : [CMS_{ID}, TNS_{ID}, OTK_{CMS,C}, n_4])$ and $NST = NS_{token} = E(K_{CMS,NS} : [CMS_{ID}, \mathcal{MTU}, OTK_{CMS,NS})$ and goto Step SSTG3. *CMS* issues a random nonce namely $OTK_{CMS,C}$ for NS_j and encapsulate the same into NST wherein *CMS* computes another random nonce $OTK_{CMS,C} = h(OTK_{CMS,NS} || SID^j_{NS})$ and enclose it into CT.

Step SSTG3: *CMS* sends a message as $MC_6 = \{\mathcal{MTU}, TNS_{ID}, C_{token}, NS_{token}\}$ to C_i by acknowledging C_i's request message (MC_5) via a public channel.

Step SSTG4: Upon receiving the message MC_6, C_i checks both nonces (that is $n_4 \in \{Z_7, C_{token}\}$) are equals or not (see Figure 18). If both are equals then C_i accepts *CMS*'s response else, rejects *CMS*'s response.

Finally, after getting both the tokens (CT and NST) from *CMS*, HCA_j initiates a session key agreement process with the Namenode Server (NS_j) followed by a mutual authentication process discussed below.

8) SESSION KEY AGREEMENT WITH SERVICE SERVER

After receiving the service server token (NS_{token}) from *CMS*, HCA_j initiate a session key agreement between C_i and NS_j. A diagram summarizing the communication message exchanges between C_i and NS_j involved throughout the session key agreement process is shown in Figure 18, and it contains the following steps:

Step SKABSSA1: HCA_j decrypts the C_{token} and extracts $OTK_{CMS,C}$. HCA_j computes a modified signature pair as $\{\psi^1_{C_i}, V_{cc}\}$ using *cModifiedSignature*$(\cdot)$ function, where $V_{cc} = R_C \cdot \mathbb{H}_2(\mathbb{H}_1(\mathcal{MTU} || OTK_{CMS,C})) + S_C \cdot \mathbb{H}_2(\psi^1_{C_i}) \pmod q$ (see Figure 19) and goto Step SKABSSA2.

Step SKABSSA2: HCA_j constructs a message $MC_7 = \{\mathcal{MTU}, TNS_{ID}, \psi^1_{C_i}, V_{cc}, NS_{token}, Cert_C\}$ and send the same to NS_j via a public channel.

Step SKABSSA3: After getting MC_7, NS_j decrypts NS_{token} and check $\mathcal{MTU}' \stackrel{?}{=} \mathcal{MTU}$ utilizing *ccmsVerif*$(\cdot)$ function (see Figure 20). If it verifies successfully then NS_j understands the legitimacy of client C_i and goto Step SKABSSA4; otherwise, reject C_i's request.

Step SKABSSA4: NS_j computes $OTK_{CMS,C} = h(OTK_{CMS,NS} || SID^j_{NS})$ and verifies the legitimacy of both C_i and *CMS* by checking the following condition: $V_{cc} \cdot \mathcal{G} = \psi^1_{C_i} \cdot \mathbb{H}_2(\mathbb{H}_1(\mathcal{MTU} || OTK_{CMS,C})) + Q_C \cdot \mathbb{H}_2(\psi^1_{C_i})$ (see Figure 20). If the condition is satisfied then NS_j realizes the legitimacy of client C_i and computes

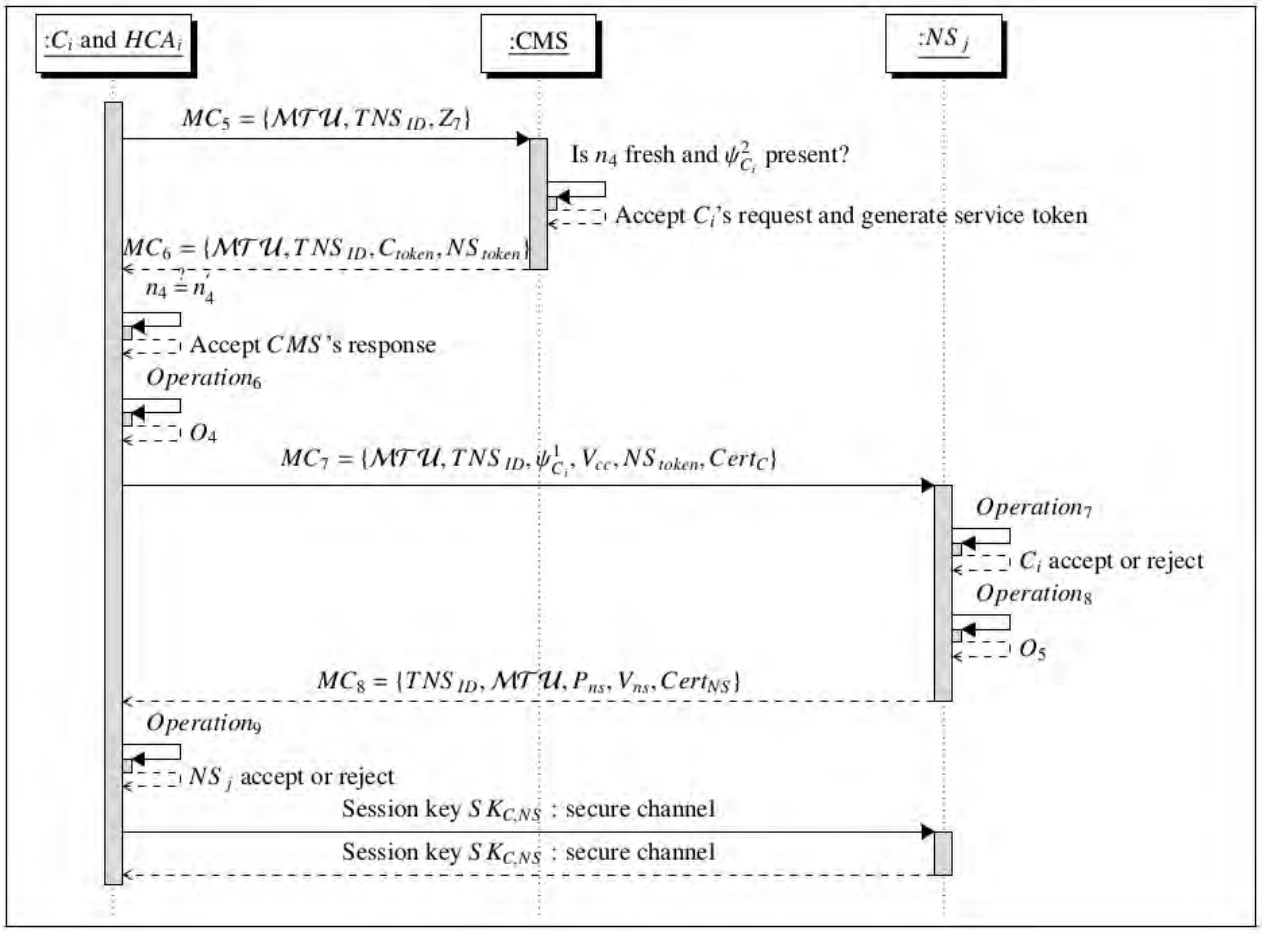

FIGURE 18. Summary of service ticket granting and session key agreement." Note: Here, $C_{token} = E(SK_{(C,CMS)} : [CMS_{ID}, TNS_{ID}, OTK_{CMS,C}, n_4])$, $Z_7 = E(SK_{C,CMS} : [\mathcal{MTU}, n_4, \psi^2_{C_i}])$, $NS_{token} = E(K_{CMS,NS} : [CMS_{ID}, \mathcal{MTU}, OTK_{CMS,NS})$, $OTK_{CMS,C} = h(OTK_{CMS,NS} || SID^j_{NS})$, $Operation_6$, $Operation_7$, $Operation_8$, and $Operation_9$: signifies the execution of the function *cModifiedSignature(·)*, *ccmsVerif(·)*, *nsSigPairGen[ ](·)* and *nscmsVerif(·)* (refer Figure 19, Figure 20, Figure 21 and Figure 22), O_4 and O_5: denotes the outcome of *cModifiedSignature(·)* and *nsSigPairGen[ ](·)* functions and $OTK_{CMS,NS}$ is a fresh nonce issued by *CMS*.

function *cModifiedSignature*$(OTK_{CMS,C}, S_C)$
{
Input: CMS's one-time key, C_i's private key
Output: C_i's modified signature pair
Domain Parameters: $\mathbb{H}_2(\cdot), \mathbb{H}_1(\cdot), \mathcal{G}, q$
Data: $\psi^1_{C_i}, \psi^2_{C_i}, TNS_{ID}, \mathcal{MTU}, NS_{token}, Cert_C, R_C$
1. Compute $V_{cc} := R_C \cdot \mathbb{H}_2(\mathbb{H}_1(\mathcal{MTU}||OTK_{CMS,C})) + S_C \cdot \mathbb{H}_2(\psi^1_{C_i}) \pmod q$ [k]
2. return $\psi^1_{C_i}, V_{cc}$
}

[k]Note: Here, C_i digitally sign on $\mathcal{MTU}, OTK_{CMS,C}$ and $\psi^1_{C_i}$ using its private key S_C, where $OTK_{CMS,C} = h(OTK_{CMS,NS}||SID^j_{NS})$.

FIGURE 19. Signature updation process at C_i's workstation.

a session key $SK_{C,NS} = h(R_{NS} \cdot \psi^1_{C_i}||OTK_{CMS,C}) = h(R_{NS} \cdot R_C \cdot \mathcal{G}||OTK_{CMS,C})$, and goto Step SKABSSA5, else reject C_i's request.

Step SKABSSA5: NS_j computes a signature pair as $\{P_{ns}, V_{ns}\}$ using *nsSigPairGen[ ](·)* (see Figure 21). NS_j constructs a message $MC_8 = \{TNS_{ID}, \mathcal{MTU}, P_{ns}, V_{ns}, Cert_{NS}\}$ and goto Step SKABSSA6.

Step SKABSSA6: NS_j sends MC_8 to C_i via a public channel.

Step SKABSSA7: Upon receiving the message MC_8, C_i checks the legitimacy of both NS_j and CMS by substantiating the following condition: $V_{ns} \cdot \mathcal{G} = P_{ns} \cdot \mathbb{H}_2(\mathbb{H}_1(TNS_{ID}||OTK_{CMS,C})) + Q_{NS} \cdot \mathbb{H}_2(P_{ns})$ (see *nscmsVerif(·)* function in Figure 22). If the condition is satisfied then C_i realizes the legitimacy of both service server NS_j and CMS, and computes a session key $SK_{C,NS} = h(R_C \cdot P_{ns}||OTK_{CMS,C}) = h(R_C \cdot R_{NS} \cdot \mathcal{G}||OTK_{CMS,C})$, and goto Step SKABSSA8, otherwise; reject C_i's request.

After establishment of the session key, HCA_j redirects client C_i to HDFS Client application instance say $HDCL_j$ to store (fresh write operation) or append or read Big Data under the supervision of NS_j. By utilizing the aforesaid process, C_i could establish a secure channel with JT_j for processing Big Data.

Proof of Correctness: In order to verify the legitimacy of both C_i and CMS, NS_j needs to check $V_{cc} \cdot \mathcal{G} = \psi^1_{C_i} \cdot$

function *ccmsVerif*($NS_{token}, \psi^1_{C_i}, V_{cc}, Cert_C$)
{
Input: CMS's issued authorization token, Modified signature pair of C_i, C_i's certificate
Output: Accept or Reject
Domain Parameters: $\mathbb{H}_2(\cdot), \mathbb{H}_1(\cdot), \mathcal{G}, q$
Data: $K_{CMS,NS}, TNS_{ID}, SID^j_{NS}$
1. Retrieve Q_C from $Cert_C$
/* C_i's public key obtained from C_i's certificate */
2. Retrieve $OTK_{CMS,NS}$ from NS_{token}
/* Decrypt NS_{token} using $CMS-NS$'s shared secret key */
3. Compute $OTK_{CMS,C} := h(OTK_{CMS,NS}||SID^j_{NS})$
4. Compute $Temp^{NS}_1 := V_{cc} \cdot \mathcal{G}$
5. Compute $Temp^{NS}_2 := \psi^1_{C_i} \cdot \mathbb{H}_2(\mathbb{H}_1(\mathcal{MTU}||OTK_{CMS,C})) + Q_C \cdot \mathbb{H}_2(\psi^1_{C_i})$
if($\mathcal{MTU}' = \mathcal{MTU}$ && $Temp^{NS}_1 = Temp^{NS}_2$)
{
6. Choose a pseudo-random number $R_{NS} \in \mathbb{Z}^*_q$
7. Compute $SK := h(R_{NS} \cdot \psi^1_{C_i}||OTK_{CMS,C}) := h(R_{NS} \cdot R_C \cdot \mathcal{G}||OTK_{CMS,C})$
8. return Accept // accept C_i and CMS //
}
else
{
9. return Reject // reject C_i and CMS //
}
}

FIGURE 20. C_i and *CMS* verification and key formation at NS_j server.

function *nsSigPairGen[ ]*($TNS_{ID}, S_{NS}, OTK_{CMS,C}$)
{
Input: NS_j's masked identity, NS_j's private key, CMS's issued random nonce
Output: Two ECC points as P_{ns} and V_{ns}
Domain Parameters: $\mathbb{H}_2(\cdot), \mathbb{H}_1(\cdot), \mathcal{G}, q$
Data: $TNS_{ID}, Cert_{NS}, R_{NS}$
1. Compute $P_{ns} := R_{NS} \cdot \mathcal{G}$
2. Compute $V_{ns} := R_{NS} \cdot \mathbb{H}_2(\mathbb{H}_1(TNS_{ID}||OTK_{CMS,C})) + S_{NS} \cdot \mathbb{H}_2(P_{ns}) \pmod q$ [m]
3. return P_{ns}, V_{ns}
}

[m]Here, NS_j digitally sign on both $TNS_{ID}, OTK_{CMC,C}$ and P_{ns} using its private key S_{NS}.

FIGURE 21. Signature pair generation process at NS_j's server.

function *nscmsVerif*($C_{token}, P_{ns}, V_{ns}, Cert_{NS}$)
{
Input: CMS's issued authorization token, signature pair of NS_j, NS_j's certificate
Output: Accept or Reject
Domain Parameters: $\mathbb{H}_2(\cdot), \mathbb{H}_1(\cdot), \mathcal{G}, q$
Data: $TNS_{ID}, \mathcal{MTU}, OTK_{CMS,C}$
1. Retrieve Q_{NS} from $Cert_{NS}$
/* NS_j's public key obtained from NS_j's certificate */
2. Compute $Temp^{C'}_1 := V_{ns} \cdot \mathcal{G}$
3. Compute $Temp^{C'}_2 := P_{ns} \cdot \mathbb{H}_2(\mathbb{H}_1(TNS_{ID}||OTK_{CMS,C})) + Q_{NS} \cdot \mathbb{H}_2(P_{ns})$
if($Temp^{C'}_1 = Temp^{C'}_2$)
{
4. Compute $SK := h(R_C \cdot P_{ns}||OTK_{CMS,C}) := h(R_C \cdot R_{NS} \cdot \mathcal{G}||OTK_{CMS,C})$
5. return Accept // accept NS_j and CMS //
}
else
{
6. return Reject // reject NS_j and CMS //
}
}

FIGURE 22. NS_j's signature verification and key formation at C_i's workstation.

$\mathbb{H}_2(\mathbb{H}_1(\mathcal{MTU}\ ||OTK_{CMS,C}))+ Q_C \cdot \mathbb{H}_2(\psi^1_{C_i})$. To satisfy the verification condition, it must holds $\psi^1_{C_i} = R_C \cdot \mathcal{G}$ and $Q_C = S_C \cdot \mathcal{G}$. Similarly, to verify the legitimacy of both NS_j and CMS, C_i needs to verify $V_{ns} \cdot \mathcal{G} = P_{ns} \cdot \mathbb{H}_2(\mathbb{H}_1(TNS_{ID}||OTK_{CMS,C})) + Q_{NS} \cdot \mathbb{H}_2(P_{ns})$. To satisfy the condition, it must satisfies $P_{ns} = R_{NS} \cdot \mathcal{G}$ and $Q_{NS} = S_{NS} \cdot \mathcal{G}$.

9) SECURE AND INTEGRITY-ASSISTED WRITE OR APPEND OPERATION IN HDFS

Suppose C_i and NS_j are having the session key $SK_{C,NS}$ between themselves. Now, say for instance, C_i has four files (say f_1, f_2, f_3 and f_4) and each file is having 250 GB (gigabytes) data. C_i wants to import these files from other external sources into HDFS, and it needs to create a Big File say F_1 with the file size of 1 TB (terabytes). According to the policy of HDFS, the file F_1 should go through the HDFS-Write task (see the basic HDFS-Write operation in Figure 23).

In this regard, NS_j assigns several chunk servers or Datanodes to HDFS Client ($HDCL_j$) for data streaming via secure channel. In such a provision, $HDCL_j$ divides the file F_1 into $\frac{1024 \times 1024\ MB}{64\ MB} = 16384$ number of blocks and write these blocks into several Datanodes along with the replicas (two replicas for each block). Before writing each block (i^{th} block) including its replica into the chunk servers, $HDCL_j$ computes $HMAC^i_{BL_{ID}} = h_{(K_{NMS,C})}(C_{ID}||BL^i_{ID}\ ||h(BL^i_{data}))$ of i^{th} block, where BL^i_{ID} represents the identity of i^{th} block and BL^i_{data} denotes the data content (64 MB data) of the i^{th} block. $HDCL_j$ then stores both the i^{th} block and $HMAC^i_{BL_{ID}}$ to the respective chunk server for future activities (namely integrity-assisted read and processing the Big Data).

If client C_i needs to append additional data in the existing file F_1 in the near future, he needs to follow the similar strategy as write operation as discussed above.

10) SECURE INTEGRITY-ASSISTED READ IN HDFS

In this phase, C_i wants to read his previously archived Big Data (here the file F_1) from HDFS. The basic work-flow of HDFS-Read operation is shown in Figure 24. According to the policy of HDFS, after getting the block locations from NS_j via a secure channel followed by a session key ($SK_{C,NS}$) establishment process, C_i can directly read the blocks from the chunk servers or Datanode servers through $HDCL_j$. Suppose C_i needs to read the i^{th} block of the file F_1. In this connection, $HDCL_j$ loads the same block and computes $HMAC^{i'}_{BL_{ID}} = h_{(K_{NMS,C})}(C_{ID}||BL^i_{ID}\ ||h(BL^{i'}_{data}))$. After that, $HDCL_j$ verifies that $HMAC^{i'}_{BL_{ID}} \stackrel{?}{=} HMAC^i_{BL_{ID}}$. If the condition is satisfied then C_i realizes the data integrity of i^{th} block. Similarly, after verifying the data integrity of all the blocks belongs to F_1 (here, F_1 consists of 16384 blocks), C_i set a boolean variable as $flag$ = "true" which indicates that the file F_1 is not being modified by any attackers or insiders (more precisely, the data integrity of file F_1 has been preserved), otherwise; C_i set the $flag$ = "$false$". If the $flag$ variable returns false for any j^{th} block then C_i goes through the service provider's Service Level Agreement (SLA) and takes proper action, else C_i outsources map-reduce code [76] and $flag$ to JT_j for Big Data processing as follows.

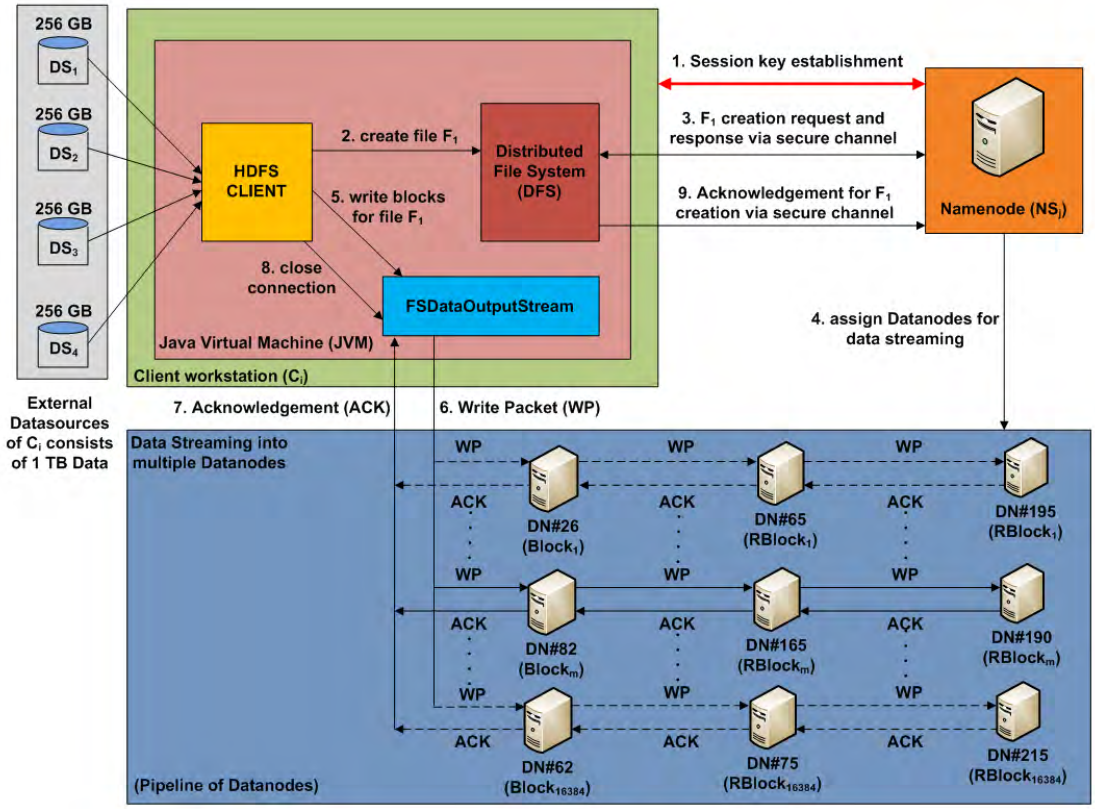

FIGURE 23. Summary of HDFS-Write operation.

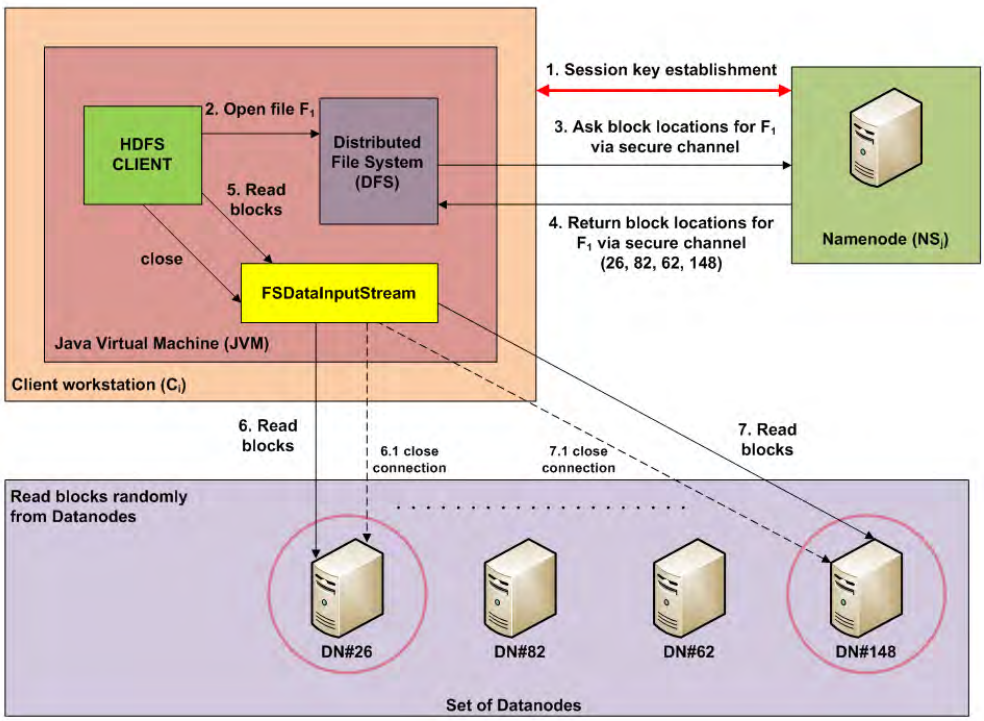

FIGURE 24. Summary of HDFS-Read operation.

11) SECURE BIG DATA PROCESSING USING MAPREDUCE

At the beginning of this phase, C_i establishes a session key (say $SK_{C,JT}$) with JT_j followed by a mutual authentication and key agreement process (see Sec. V-D.8). After the key formation, C_i sends the map-reduce code [76] and the *flag* to JT_j via secure channel. After getting these information,

JT_j assigns different Task Trackers (TTs) or Datanodes to execute the map-reduce code for different blocks of C_i's file (say F_1). After successful execution of the code, JT_j stores the result into C_i's local filesystem in encrypted format utilizing the key $SK_{C,JT}$.

12) PASSWORD CHANGE PHASE

This section discuss about the password updation process of client C_i and *BDSP*'s administrator. Assume, the client C_i needs to change his password for security reasons. To do this, the following steps need to be executed. Note here, after the execution of user login phase (refer Sec. V-D.5), HCA_j has the following information: $R_{C_i}^{CMS^*} = R_{C_i}^{CMS} \oplus h(r_i \,||PWD)$, $\mathcal{MTU} = h(C_{ID}||r_i \,||HCA_{ID}||\, R_{C_i}^{CMS})$, $r_i^* = r_i \oplus h(C_{ID} \,||\, PWD)$, $K_{(NMS,C)}^* = K_{(NMS,C)} \oplus h(PWD \,||\, R_{C_i}^{CMS})$ and $USPW_i = h(C_{ID} \,||HCA_{ID} \,||PWD \,||r_i)$.

Step PCP1: C_i enters his identity C_{ID} and old password P^{old} into HCA_j and goto Step PCP2.

Step PCP2: HCA_j locally verifies the condition $UPW_i' \stackrel{?}{=} USPW_i$, where $UPW_i' = h(C_{ID} \,||HCA_{ID}||\, P^{old} \,||r_i)$ and $r_i = r_i^* \oplus\ h(C_{ID} \,||\, PWD^{old})$. If the condition is satisfied then HCA_j asks C_i to enter his updated password and goto Step PCP3, otherwise; re-enter the user id. and password again in HCA_j.

Step PCP3: C_i enters a new password P^{new} into HCA_j. HCA_j selects a random number r_i^{new} and computes $P^{new'} = h(h(r_i^{new} \,||P^{new})||R_{C_i}^{CMS})$, where $R_{C_i}^{CMS} = R_{C_i}^{CMS^*} \oplus h(r_i \,||PWD)$, and goto Step PCP4.

Step PCP4: HCA_j computes $\mathcal{U}_{Dtl}'' = E(K_{(NMS,C)} : [P^{new'}, \mathcal{MTU}])$ where $K_{(NMS,C)} = K_{(NMS,C)}^* \oplus h(PWD \,||\, R_{C_i}^{CMS})$. HCA_j select a random nonce n_5 and constructs a password update message as Pwd_chg_msg $= \{CMS_{ID}, \mathcal{MTU}, n_5, \mathcal{U}_{Dtl}''\}$ (it is similar as the message msg_{C5} shown in Figure 11) and goto Step PCP5.

Step PCP5: HCA_j sends Pwd_chg_msg to *CMS*.

Step PCP6: After getting Pwd_chg_msg, *CMS* checks $\mathcal{MTU}$ is an existing user or not. If user presents then goto Step PCP7, otherwise; reject C_i's password change request.

Step PCP7: *CMS* broadcasts both $\mathcal{MTU}$ and $\mathcal{U}_{Dtl}''$ (similar as the message $msg_{C5.1}$ shown in Figure 11) to *ES* and *NMS*. After decrypting $\mathcal{U}_{Dtl}''$, both *ES* and *NMS* updates their user databases and broadcasts their acknowledgement messages (same as the message $msg_{C5.2}$ shown in Figure 11) to *CMS*. Thereafter, *CMS* sends a response message (similar as the message msg_{C6} shown in Figure 11) to C_i about the confirmation of password updation request.

Step PCP8: After getting the response message from *CMS*, HCA_j computes $r_i^{new^*} = r_i^{new} \oplus h(C_{ID} \,||P^{new})$, $K_{(NMS,C)}^{new*} = K_{(NMS,C)} \oplus h(P^{new} \,||\, R_{C_i}^{CMS})$ and $USPW_i^{new} = h(C_{ID} \,||HCA_{ID} \,||P^{new} \,||r_i^{new})$, respectively and stores them in its database for future use.

Note that using the aforesaid mechanism, *BDSP*'s administrator (*BA*) can update its password for security reasons utilizing both *NMS* and *ES* servers and the service provider application HSA_k.

Remark 3: In order to prevent replay attack between C_i and NS_j, NS_j can temporarily keep $\psi_{C_i}^1$ and V_{cc} in its system cache. The similar mechanism has been reported in [18] and [10] can be applied for the replay attack protection. Suppose a similar message $MC_7' = \{\mathcal{MTU}, TNS_{ID}, \psi_{C_i}^{1'}, V_{cc}', NS_{token}, Cert_C\}$ has been received by NS_j. NS_j first checks $\psi_{C_i}^{1'} = \psi_{C_i}^1$ and $V_{cc}' = V_{cc}$. If both the conditions are satisfied successfully then NS_j treats the message MC_7' as replay message, otherwise; NS_j treats the message MC_7' is fresh and NS_j updates $\psi_{C_i}^1$ and V_{cc} with $\psi_{C_i}^{1'}$ and V_{cc}' in its system cache. In a similar way, the same procedure can be applied for other phases to protect replay attack.

Remark 4: To proof the proposed HEAP-KDC is fault-tolerant in terms of secret credentials storage, we distribute the credentials of clients (C_i), service provider (*BDSP*) and service servers (NS_j and JT_j) among different servers in the following settings:

1) *CMS*: It is permissible to store the masked identities ($\mathcal{MTU}$'s, TNS_{ID}s and TJT_{ID}s), masked passwords (SID_{NS}^js, SID_{JT}^js and $BDSP_{PWD}^*$s), pass-phrases ($K_{CMS,BDSP}$s) secret keys ($K_{CMS,BDSP}$s, $K_{CMS,NS}$s, $K_{CMS,JT}$s) of service providers and service servers, the dictionary of password transformation parameters of clients ($R_{C_i}^{CMS}$s), and the masked identities of clients ($\mathcal{MTU}$s), respectively.
2) *NMS*: This server keeps the masked identities, pass-phrases and passwords of clients ($\mathcal{MTU}$s, $K_{NMS,C}$s and PWD^*), and the dictionary of password transformation parameters of both service providers and service servers (R_{BDSP}^{NMS}s, $R_{NS_j}^{NMS}$s and $R_{JT_j}^{NMS}$), respectively.
3) *ES*: It keeps all the secret credentials of both *CMS* and *NMS* servers. In fact, at the time of online registration process, both *CMS* and *NMS* send their principal's secret credentials to *ES*. Note here, *ES* is a fully trusted server since it is not reachable online to any other principals (except *CMS* and *NMS*) including adversaries at the time of authentication process.

In such a settings, if *CMS* (or *NMS*) fails due to some hardware failure and the server data is totally lost then it could be rebuild from *ES* server. Similarly, if *ES* server fails then the entire legacy data can be extracted from both *CMS* and *NMS* servers to renovate a new *ES*. Thus, we can remark that for the current settings of HEAP-KDC, the proposed authentication framework is more dependable and fault-tolerant.

Remark 5: At the time of mutual authentication and single sign-on process between C_i and *CMS*, suppose any one of the random nonce R_C or R_{CMS} is known to the adversary $\mathcal{A}dv$. In this connection it is obvious that $\mathcal{A}dv$ can compute $R_C \cdot \pi_{CMS}^1$ or $R_{CMS} \cdot \psi_{C_i}^1$, respectively. In such a provision, to protect known session-specific temporary information attacks from $\mathcal{A}dv$ both C_i and *CMS* can compute the session key as $SK_{C,CMS} = h(R_C \cdot \pi_{CMS}^1 || R_{C_i}^{CMS}) = h(R_{CMS} \cdot \psi_{C_i}^1 || R_{C_i}^{CMS})$.

Therefore, the adversary $\mathcal{A}dv$ requires the knowledge of $R_{C_i}^{CMS}$ to compute the actual session key between C_i and CMS. In the same way, $BDSP$ and NMS can compute the sesseion key $SK_{BDSP,NMS}$ utilizing R_{BDSP}^{NMS}.

VI. SECURITY ANALYSIS

To prove that the proposed protocol (HEAP) is provably secure, we analyze the security of HEAP by utilizing both formal and informal security analysis. Further, to pursue the security analysis, we also use the proposed threat model as discussed in Section V-B.

A. FORMAL SECURITY ANALYSIS USING ROR MODEL

In order to present formal security analysis of HEAP in detail, we first discuss few terminologies of the widely-used Real-Or-Random (ROR) model [77]. Thereafter, utilizing the same model we substantiate that the proposed protocol (HEAP) provides the session-key (SK) security against an adversary $\mathcal{A}dv$ in Theorem 1.

1) ROR MODEL

In HEAP, we have four entities, namely, user C_i, CMS, NMS and NS_j (or Big Data processing service server JT_j). The following attributes are involved in the ROR model.

a: PARTICIPANTS

Suppose $\Gamma_{C_i}^{s}$, Γ_{CMS}^{t}, Γ_{NMS}^{u} and $\Gamma_{NS_j}^{v}$ (or $\Gamma_{JT_j}^{w}$) are the instances s, t, u and v (or w) of the principals C_i, CMS, NMS and NS_j (or JT_j), respectively. These instances are also coined as the oracles.

b: ACCEPTED STATE

An instance, say Γ^s is said to be in accepted state, if it reaches to an accept state after receiving the last protocol message. The session identification (sid) is constructed by concatenating all the communicated messages (sent and received messages) of the Γ^s for the current executing session.

c: PARTNERING

We say two instances Γ^s and Γ^t are partnered to each other if the following three conditions are satisfied simultaneously: 1) both Γ^s and Γ^t are in accept state, 2) both Γ^s and Γ^t mutually authenticate each other and also share the same sid, and 3) both Γ^s and Γ^t are mutual partners of each other.

d: FRESHNESS

If the established session key SK between C_i and NS_j (or between C_i and JT_j) is not disclosed using the following $Reveal(\Gamma^s)$ query, $\Gamma_{C_i}^{s}$ or $\Gamma_{NS_j}^{v}$ (or $\Gamma_{JT_j}^{w}$) is said to be fresh.

e: ADVERSARY

In ROR assumptions, all the message communications can be supervised by $\mathcal{A}dv$ including eavesdropping, modifying, deleting, and inserting transmitted messages. In addition, $\mathcal{A}dv$ can have the access of the following queries [78]:

1) $Execute(\Gamma^s, \Gamma^v)$ – A *passive attack* is modeled utilizing this query wherein $\mathcal{A}dv$ can have access to the transmitted messages between two legitimate parties.
2) $Send(\Gamma^s, M)$ – This query is modeled as an *active attack*, wherein a message, say M can be transmitted to a participant instance, say Γ^s and also receives a response message.
3) $Reveal(\Gamma^s)$ – An adversary $\mathcal{A}dv$ discloses the current session key SK of Γ^s (and its partner) utilizing this query.
4) $CorruptSmartWorkstation(\Gamma_{SW_i}^{s})$ – This query represents an active attack wherein HEAP-KDC's authentication token (Z_4') and C_i's authorization tokens (C_{token} and NS_{token}) are leaked to the adversary $\mathcal{A}dv$ by compromising C_i's workstation. These information leakage is modeled using this query to check the security of the proposed protocol. It is reported in [61] that $CorruptSmartWorkstation(\cdot)$ query fortifies the weak-corruption model, where the temporary keys and internal credentials related to the participant instances are not corrupted.
5) $Test(\Gamma^s)$ – Applying this query, the semantic security of the session key SK is being modeled adopting the indistinguishability in ROR [77]. Before starting the experiment, an unbiased coin cn needs to be flipped and its result is only known to $\mathcal{A}dv$. This result decides the output of the $Test$ query. If $\mathcal{A}dv$ executes this query, and also SK is fresh, Γ^s outputs SK when $cn = 1$ or a random number in the same domain when $cn = 0$; otherwise, it will produce the output as a null value ($\perp$).

f: SEMANTIC SECURITY OF SESSION KEY

Under the ROR assumptions, $\mathcal{A}dv$ needs to apprehend a participant instance's real session key from a random key. To achieve this purpose, $\mathcal{A}dv$ can execute several $Test$ queries against either $\Gamma_{C_i}^{s}$ or $\Gamma_{NS_j}^{v}$ (or $\Gamma_{JT_j}^{w}$). At the end of this experiment, $\mathcal{A}dv$ guesses a bit cn', and he or she wins the game if $cn' = cn$. Assume $\mathcal{S}$ is an event that $\mathcal{A}dv$ can win the game, $\mathcal{A}dv$'s advantage in breaking the semantic security of the proposed protocol (HEAP) is denoted and defined by $Adv_{HEAP}^{AKE} = |2 \cdot Pr[\mathcal{S}] - 1|$. If $Adv_{HEAP}^{AKE} \leq \epsilon$, for a sufficiently small $\epsilon > 0$, we say HEAP provides SK-security.

g: RANDOM ORACLE

Let us assume that the cryptographic one-way hash function $h(\cdot)$ is available to all the entities including $\mathcal{A}dv$. We model $h(\cdot)$ by a random oracle, say $\mathcal{H}$ [10], [78].

2) SECURITY PROOF

Theorem 1 substantiates the semantic security of the proposed protocol (HEAP) under ROR model.

Theorem 1: Let $\mathcal{A}dv$ be an adversary running in polynomial time t against the proposed authentication protocol, $HEAP$ in the ROR model, and $\mathcal{D}$, q_h, q_s, $|\mathcal{H}|$, $|\mathcal{D}|$, $Adv_{\mathcal{A}dv}^{ECDDHP}(t)$ and $Adv_{\mathcal{E}}^{IND-CPA}(k)$ denote the uniformly

distributed password dictionary, the number of *hash* queries, $Send(\cdot)$ queries, the range space of $h(\cdot)$, the length of $\mathcal{D}$, the advantage of $\mathcal{A}dv$ in breaking ECDDHP and the advantage of $\mathcal{A}dv$ in breaking the IND-CPA secure symmetric cipher $\mathcal{E}$ (provided in Definition 1), respectively, and $Adv_{\mathcal{E}}^{IND-CPA}(K) = Adv_{\mathcal{E},SGL}^{IND-CPA}(K)$ or $Adv_{\mathcal{E},MEL}^{IND-CPA}(K)$. Then, $\mathcal{A}dv$'s advantage in breaking the semantic security of HEAP can be estimated as

$$Adv_{HEAP}^{AKE} \leq \frac{q_h^2}{|\mathcal{H}|} + 2\Big(\frac{q_s}{|\mathcal{D}|} + Adv_{\mathcal{A}dv}^{ECDDHP}(t) + Adv_{\mathcal{E}}^{IND-CPA}(K)\Big).$$

Proof: In order to proof the above theorem, we go through a sequence of six games, say $\mathcal{GM}_j$ ($j = 0, 1, 2, 3, 4, 5$) as in [19] and [78]. Let the initial game be $\mathcal{GM}_0$ and the final game be $\mathcal{GM}_5$. Assume that $\mathcal{S}_i$ represents as an event wherein $\mathcal{A}dv$ can successfully guess the bit cn in the $Test(\cdot)$ query with respect to the game $\mathcal{GM}_j$. All the games are outlined as follows.

- Game $\mathcal{GM}_0$: This is the initial game where the adversary $\mathcal{A}dv$ incorporates a real attack against HEAP under ROR model. Before starting of the game $\mathcal{GM}_0$, $\mathcal{A}dv$ chooses a bit cn. Under the ROR assumptions, the initial game $\mathcal{GM}_0$ and the actual protocol are identical to each other. Hence, it follows that
$$Adv_{HEAP}^{AKE} = |2 \cdot Pr[\mathcal{S}_0] - 1|. \quad (1)$$
- Game $\mathcal{GM}_1$: In this game, $\mathcal{A}dv$ performs the eavesdropping attacks by running the $Execute(\cdot)$ query. Finally, at the end of this game, $\mathcal{A}dv$ needs to call the $Test(\cdot)$ query. The result of the $Test(\cdot)$ query determines whether $\mathcal{A}dv$ obtains the real session key SK or a random number. In HEAP, the session key between C_i and NS_j is computed as $SK_{C,NS} = h(R_C \cdot P_{ns} \,||OTK_{CMS,C}) = h(R_C \cdot R_{NS} \cdot \mathcal{G} \,||OTK_{CMS,C})$. Therefore, to reveal the session key $SK_{C,NS}$, $\mathcal{A}dv$ needs the knowledge about two random secrets R_C and R_{NS}. Since we encapsulated these two secrets inside the exchanged messages (see MC_7 and MC_8 in Section V-D.8) indirectly, eavesdropping attacks against these messages are not beneficial to determine $SK_{C,NS}$. Therefore, the probability of winning $\mathcal{GM}_1$ by incorporating eavesdropping attacks by $\mathcal{A}dv$ is negligible. As a result, we say that both the games $\mathcal{GM}_0$ and $\mathcal{GM}_1$ are indistinguishable. Thus, we infer that
$$Pr[\mathcal{S}_1] = Pr[\mathcal{S}_0]. \quad (2)$$
- Game $\mathcal{GM}_2$: We include the simulation of both $Send(\cdot)$ and $\mathcal{H}(\cdot)$ queries into this game and transform the game $\mathcal{GM}_1$ to the game $\mathcal{GM}_2$. This game is also modeled as an active attack. In this game, $\mathcal{A}dv$ eavesdrops all the exchanged messages $(MC_1, MC_2, \cdots, MC_8)$. According to the policy of HEAP, we appended a random nonce with each communicating message (send and receive) so that there will be no collision of hash outputs when $\mathcal{A}dv$ simulates it utilizing the $Send(\cdot)$ query. Thus, the birthday paradox results in the following inequality:
$$|Pr[\mathcal{S}_2] - Pr[\mathcal{S}_1]| \leq \frac{q_h^2}{2|\mathcal{H}|}. \quad (3)$$
- Game $\mathcal{GM}_3$: This game is modeled as an active attack wherein $\mathcal{A}dv$ tries to compute the current session key $SK_{C,NS}$ between C_i and NS_j by obtaining the other credentials (specifically, V_{cnew} and $\mathcal{MTU}$) from the guessed password (PWD) of C_i and the game $\mathcal{GM}_2$ is transformed into the game $\mathcal{GM}_3$. Suppose $\mathcal{A}dv$ eavesdrops all the exchanged messages MC_i ($i = 1, 2, \cdots, 8$) of the current session. We utilize C_i's transformed password ($PWD^* = h(h(r_i|| \, PWD)|| \, R_{C_i}^{CMS})$) only in order to encrypt the CMS's identifier (i.e., V_{cnew}) during the single sign-on and session key establishment process, but $\mathcal{A}dv$ does not have any provisions to check the transformed password of C_i directly on the server-sides (both CMS and NS_j). Therefore, even if $\mathcal{A}dv$ guesses C_i's password, but he or she has no scope to verify it on the server-side. Moreover, to reveal the actual password (PWD) of C_i correctly from client-end, $\mathcal{A}dv$ requires the knowledge about $R_{C_i}^{CMS}$ and r_i. In fact, if the authentication system has a provision to check the limited number of incorrect passwords as inputs, we have the following result:
$$|Pr[\mathcal{S}_3] - Pr[\mathcal{S}_2]| \leq \frac{q_s}{|\mathcal{D}|}. \quad (4)$$
- Game $\mathcal{GM}_4$: This game is imitated as an active attack and the game $\mathcal{GM}_3$ is transformed into the game $\mathcal{GM}_4$. In this game, $\mathcal{A}dv$ tries to compute the session key $SK_{C,NS}$ by utilizing both the public information ($\psi_{C_i}^1 = R_C \cdot \mathcal{G}$ and $P_{ns} = R_{NS} \cdot \mathcal{G}$) of C_i and NS_j and previously eavesdropped messages from the aforesaid discussed games. Since both C_i and NS_j can compute the session key as $SK_{C,NS} = h(R_C \cdot P_{ns} \,||OTK_{CMS,C}) = h(R_C \cdot R_{NS} \cdot \mathcal{G} \,||OTK_{CMS,C})$ by utilizing CMS server, it is obvious that having the knowledge about $\psi_{C_i}^1$ and P_{ns}, it is computationally hard to derive $SK_{C,NS}$ due to the difficulty of solving ECDDHP (see Section IV). Therefore, it follows that
$$|Pr[\mathcal{S}_4] - Pr[\mathcal{S}_3]| \leq Adv_{\mathcal{A}}^{ECDDHP}(t). \quad (5)$$
- Game $\mathcal{GM}_5$: This is the final game and it is modeled as an active attack, and the game $\mathcal{GM}_4$ is transformed into the final game $\mathcal{GM}_5$. In this game, $\mathcal{A}dv$ compromises the C_i's workstation and tries to compute the session key $SK_{C,NS}$ by stealing the session temporal secrets, such as $OTK_{CMS,NS}$ and $OTK_{CMS,C} = h(OTK_{CMS,NS}|| \, SID_{NS}^j)$ from NS_{token} and C_{token}, respectively. But, in the proposed protocol, for encryption or decryption we use IND-CPA secure symmetric cipher, such as stateless CBC mode of AES analogy. Therefore, $\mathcal{A}dv$ requires the knowledge about $SK_{C,CMS}$ and SID_{NS}^j to decrypt both

NS_{token} and C_{token} to get the parameters $OTK_{CMS,NS}$ and $OTK_{CMS,C}$. Thus, it follows that

$$|Pr[\mathcal{S}_5] - Pr[\mathcal{S}_4]| \leq Adv_{\mathcal{E}}^{IND-CPA}(K). \quad (6)$$

Since all the queries are successfully simulated in the final game $\mathcal{GM}_5$, $\mathcal{A}dv$ is left with only guessing the bit cn for winning the game after the $Test(\cdot)$ query. Then, we have,

$$Pr[\mathcal{S}_5] = \frac{1}{2}. \quad (7)$$

From Eqs. (1) and (2), we have,

$$\begin{aligned} \frac{1}{2} \cdot Adv_{HEAP}^{AKE} &= |Pr[\mathcal{S}_0] - \frac{1}{2}| \\ &= |Pr[\mathcal{S}_1] - \frac{1}{2}|. \end{aligned} \quad (8)$$

From Eqs. (7) and (8), we have,

$$\frac{1}{2} \cdot Adv_{HEAP}^{AKE} = |Pr[\mathcal{S}_1] - Pr[\mathcal{S}_5]|. \quad (9)$$

According to the triangular inequality, we get the following:
$|Pr[\mathcal{S}_1] - Pr[\mathcal{S}_5]| \leq |Pr[\mathcal{S}_1] - Pr[\mathcal{S}_2]| + |Pr[\mathcal{S}_2] - Pr[\mathcal{S}_5]| \leq |Pr[\mathcal{S}_1] - Pr[\mathcal{S}_2]| + |Pr[\mathcal{S}_2] - Pr[\mathcal{S}_3]| + |Pr[\mathcal{S}_3] - Pr[\mathcal{S}_4]| + |Pr[\mathcal{S}_4] - Pr[\mathcal{S}_5]|$.
Now from Eqs. (3), (4), (5) and (6), we get,

$$\begin{aligned} |Pr[\mathcal{S}_1] - Pr[\mathcal{S}_5]| \leq \frac{q_h^2}{2|\mathcal{H}|} + \frac{q_s}{|\mathcal{D}|} \\ + Adv_{\mathcal{A}}^{ECDDHP}(t) \\ + Adv_{\mathcal{E}}^{IND-CPA}(K). \end{aligned} \quad (10)$$

Thus, Eqs. (9) and (10) produce the following result:

$$\begin{aligned} \frac{1}{2} \cdot Adv_{HEAP}^{AKE} &= |Pr[\mathcal{S}_1] - Pr[\mathcal{S}_5]| \\ &\leq \frac{q_h^2}{2|\mathcal{H}|} + \frac{q_s}{|\mathcal{D}|} \\ &\quad + Adv_{\mathcal{A}}^{ECDDHP}(t) \\ &\quad + Adv_{\mathcal{E}}^{IND-CPA}(K). \end{aligned} \quad (11)$$

Finally, after multiplying both sides of the above equation (Eq. 11) by a factor of 2, we get the required result as follows:

$$\begin{aligned} Adv_{HEAP}^{AKE} \leq \frac{q_h^2}{|\mathcal{H}|} + 2\Big(\frac{q_s}{|\mathcal{D}|} + Adv_{\mathcal{A}}^{ECDDHP}(t) \\ + Adv_{\mathcal{E}}^{IND-CPA}(K)\Big). \end{aligned}$$

■

B. INFORMAL SECURITY ANALYSIS

This section presents an informal security inspection of the proposed protocol (HEAP) and shows it is resilient against various other well-known attacks. This discussion are represented in the following propositions.

Proposition 1: HEAP is resilient against the privileged-insider attacks.

Proof: According to the policy of the proposed protocol, during user enrollment task HCA_j asks C_i to give his or her user identity C_{ID} and password PWD. After getting these parameters, HCA_j transforms C_i's identity and password as $\mathcal{MTU} = h(C_{ID}||r_i\ ||HCA_{ID}||\ R_{C_i}^{CMS})$ and $PWD^* = h(h(PWD||\ r_i)||R_{C_i}^{CMS})$. HCA_j encrypts both these transformed parameters and construct a message as $msg_{C5} = \{CMS_{ID}, \mathcal{MTU}, n_2, E(K_{(NMS,C)} : [PWD^*, \mathcal{MTU}, MBNO_C])\}$. HCA_j then sends the message to the CMS.

Let a privileged-insider user of the CMS, being an adversary $\mathcal{A}dv$, receives the message msg_{C5} and tries to extract the original identity of C_i from $\mathcal{MTU}$. Even if $\mathcal{A}dv$ is having the knowledge about $R_{C_i}^{CMS}$ and HCA_{ID}, but it is still not sufficient for $\mathcal{A}dv$ to trace C_i's actual identity without having the value of r_i. CMS can not decrypt the masked password $PWD^* = h(h(PWD||\ r_i)||R_{C_i}^{CMS})$ because it does not have the key $K_{NMS,C}$. Further, suppose a privileged-insider user of the NMS or ES, being an adversary $\mathcal{A}dv$, gets the transformed identity and the password of C_i and tries to extract the actual identity and password of C_i. But, the adversary $\mathcal{A}dv$ can not disclose the original identity of C_i due to the lack of knowledge about $R_{C_i}^{CMS}$ and r_i. In the same way, $\mathcal{A}dv$ can not extract C_i's original password. Thus, the proposed protocol HEAP can protect the privileged-insider attacks.

Proposition 2: HEAP protects C_i's private information against workstation compromise attacks.

Proof: Let an adversary $\mathcal{A}dv$ controls C_i's workstation after successful accomplishment of C_i's t^{th} session say S_t. In such a provision, $\mathcal{A}dv$ captures $\mathcal{MTU}$, C_{token}, NS_{token}, $\psi_{C_i}^1$, $\psi_{C_i}^2$, V_{cc}, P_{ns}, V_{ns}, Z_4' and TNS_{ID} parameters from workstation credential cache. After gaining the knowledge about these parameters, $\mathcal{A}dv$ can not derive the future session key between C_i and NS_j say $SK_{C,NS} = h(R_C^{new} \cdot P_{ns}^{new}||OTK_{CMS,C}^{new}) = h(R_{NS}^{new} \cdot \psi_{C_i}^{1new}||OTK_{CMS,C}^{new})$. Because, for the future session say S_{t+1}, C_i chooses a fresh pseudo-random number say $R_C^{new} \neq R_C$ and NS_j selects a fresh pseudo-random number say $R_{NS}^{new} \neq R_{NS}$, and CMS chooses a fresh nonce say $OTK_{CMS,C}^{new} \neq OTK_{CMS,C}^{old}$ to construct $C_{token}^{new} \neq C_{token}^{old}$. Since the adversary $\mathcal{A}dv$ does not have any knowledge about R_C^{new}, R_{NS}^{new} and $OTK_{CMS,C}^{new}$ parameters for the session say S_{t+1} then it is hard to compute the future session key.

Additionally, some information related to C_i namely $\mathcal{EI}$, r_i, $K_{NMS,C}$ and $R_{C_i}^{CMS}$ are stored into the workstation for verifying C_i's legitimacy at the time of user login and password change phase. This parameters are stored either in encrypted format or in a transformed manner using one way hash function. Therefore, without having the knowledge about the keys and breaking the hardness property of cryptographic one-way hash function, it is impossible to get the parameters. Thus, we can remark that the proposed protocol protects disclosure of C_i's confidential information through workstation compromise attacks.

Proposition 3: HEAP is resilient against denial-of-service attacks.

Proof: In order to achieve user (or service provider) login, HEAP utilizes workstation-based authentication mechanism without involving the HEAP-KDC. In this connection, C_i enters his identity (C_{ID}) and password (PWD) to HCA_j. To verify the current user C_i, HCA_j computes $\mathcal{EI}' = h(C_{ID}||PWD)$ and check it with $\mathcal{EI}$ in its cookie. If both matches then HCA_j load $r_i^* = r_i \oplus h(C_{ID}||PWD)$, $R_{C_i}^{CMS^*} = R_{C_i}^{CMS} \oplus h(r_i||PWD)$ and $USPW_i = h(C_{ID}||HCA_{ID}||PWD||r_i)$ from its database into C_i's workstation. HCA_j then computes $R_{C_i}^{CMS} = R_{C_i}^{CMS^*} \oplus \oplus h(r_i||PWD)$, $r_i = r_i^* \oplus h(C_{ID}||PWD)$ and $USPW_i^* = h(C_{ID}||HCA_{ID}||PWD||r_i)$. After that, HCA_j verifies the condition $USPW_i^* \stackrel{?}{=} USPW_i$. If it holds then HCA_j accepts C_i; otherwise, HSA_j treats C_i as illegitimate user. As the first step verification has done only on the client-side and HEAP-KDC does not involve into this process then we say that HEAP can resists server-side denial-of-service attacks.

Proposition 4: HEAP provides privacy preserving data integrity in Hadoop.

Proof: Form Proposition 8 and Assrt. 9, it is obvious that both C_i and NS_j (or JT_j) preserve their identities during session key establishment and Big Data service access task. Further, from Section V-D.9 and Section V-D.10, it could also be observe that HEAP stores (writes or appends) C_i's raw datablocks into several chunk servers (HDFS). During this process, HEAP computes a hashed MAC (HMAC) of each datablock and stores the HMAC along with the raw datablock into Datanode servers.

Suppose an adversary $\mathcal{A}dv$ or a malicious insiders change the content of the raw datablock. In such a provision, if C_i processes his Big Data (basically the intercepted and modified data content) utilizing MapReduce framework then he will not get the desired result. To overcome this problem, in our proposed protocol, at the time of auditing or reading the datablock, C_i would be able to check the integrity of each datablock utilizing his secret key ($K_{NMS,C}$) and HMAC. After checking the integrity of each datablocks, C_i is permissible to process the Big Data utilizing JT_j and it will lead C_i to get the desired output.

Proposition 5: HEAP is resilient against known session-specific temporary information attacks.

Proof: During the session key establishment process between C_i and NS_j (or JT_j), suppose any one of the random nonce R_C or R_{NS} (or R_{JT}) is known to the adversary $\mathcal{A}dv$. Therefore it is obvious that $\mathcal{A}dv$ can compute $R_C \cdot P_{ns}$ or $R_{NS} \cdot \psi_{C_i}^1$, respectively. But, it is not sufficient for the adversary $\mathcal{A}dv$ to compute the session key $SK_{C,NS} = h(R_C \cdot P_{ns}||OTK_{CMS,C}) = h(R_{NS} \cdot \psi_{C_i}^1||OTK_{CMS,C})$ without having the knowledge about $OTK_{CMS,C}$. Further to compute $OTK_{CMS,C} = h(OTK_{CMS,NS}||SID_{NS}^j)$, the adversary $\mathcal{A}dv$ also requires the knowledge of SID_{NS}^j and $OTK_{CMS,NS}$ parameters. Moreover, $\mathcal{A}dv$ is unable to extract these parameters from $OTK_{CMS,C}$ due to the one-way property of cryptographic hash function and cryptographic hardness property associated with the stateless CBC mode of AES encryption/decryption policy. It is also observed from Remark 5 that the proposed protocol alleviate the known session-specific temporary information attacks.

Proposition 6: HEAP protects man-in-the-middle attacks.

Proof: Suppose during session key establishment process, an adversary $\mathcal{A}dv$ tries to impersonate a legitimate client C_i or service server NS_j by eavesdropping the exchanged messages say MC_7 and MC_8. However, in the proposed protocol, C_i authenticates both CMS and NS_j by verifying two conditions as (1) $V_{cc} \cdot \mathcal{G} = \psi_{C_i}^1 \cdot \mathbb{H}_2(\mathbb{H}_1(\mathcal{MTU}||OTK_{CMS,C})) + Q_C \cdot \mathbb{H}_2(\psi_{C_i}^1)$ and (2) $OTK_{CMS,C} = h(OTK_{CMS,NS}||SID_{NS}^j)$, respectively utilizing both C_{token} and message MC_8. Similarly, NS_j verifies the legitimacy of both CMS and C_i by checking two conditions as (i) $V_{ns} \cdot \mathcal{G} = P_{ns} \cdot \mathbb{H}_2(\mathbb{H}_1(TNS_{ID}||OTK_{CMS,C})) + Q_{NS} \cdot \mathbb{H}_2(P_{ns})$ and (ii) $OTK_{CMS,C} = h(OTK_{CMS,NS}||SID_{NS}^j)$, respectively using NS_{token} and message MC_7. After validating the aforesaid conditions successfully, C_i (or NS_j) establishes the session key $SK_{C,NS}$ between themselves, otherwise; terminate the process. Since, the adversary $\mathcal{A}dv$ does not have the knowledge about R_C, R_{NS}, $OTK_{CMS,C}$, $OTK_{CMS,NS}$, SID_{NS}^j and $K_{CMS,NS}$ so, it is impossible to impersonate either C_i or NS_j.

In addition to this, the simulation result of AVISPA based formal verification (see Section VII) is also substantiates that the other phases of the proposed protocol is robust against man-in-the-middle attacks. Thus, we remark that the proposed protocol is resilient against man-in-the-middle attacks.

Proposition 7: HEAP is resilient against identity compromise attacks.

Proof: In this attack, to protect both C_i's and NS_j's (or JT_j) original identities (C_{ID} and TNS_{ID} (or TJT_{ID})) from an adversary say $\mathcal{A}dv$ who controls either CMS or NMS or ES, in the proposed scheme, the identities are stored in a transformed manner into those servers. For example, C_i's original identity are stored as $\mathcal{MTU} = h(C_{ID}||r_i||HCA_{ID}||R_{C_i}^{CMS})$ whereas NS_j's (or JT_j) identity are stored as $TNS_{ID} = h(NS_{ID}||HSA_{ID}||rss_1)$ (or $TJT_{ID} = h(JT_{ID}||HSA_{ID}||rss_2)$). Since, the adversary $\mathcal{A}dv$ does not have the knowledge about r_i, $R_{C_i}^{CMS}$, rss_1 and rss_2, so he cannot retrieve the original identities C_{ID}, NS_{ID} (or JT_{ID}) from $\mathcal{MTU}$ and TNS_{ID} (or TJT_{ID}), respectively. In a similar way, to protect $BDSP/BA$'s identity, it is also stored as $\mathcal{MTU}' = h(BDSP_{ID}||d||HSA_{ID}||R_{BDSP}^{NMS})$ into those servers. Thus completes the proof.

Proposition 8: HEAP supports user and Big Data service servers anonymity.

Proof: During user and service server registration process both C_i and NS_j (or JT_j) are enroled themselves with HEAP-KDC utilizing their masked identities namely $\mathcal{MTU}$ and TNS_{ID} (or TJT_{ID}). At the time of session key establishment task, C_i computes its digital signature using C_i's masked identity ($\mathcal{MTU}$) and an application generated pseudo random number (R_C). The same way, NS_j (or JT_j) encapsulates TNS_{ID} and R_{NS} to construct its digital signature. Thereafter,

the digital signature exchanges between C_i and NS_j (or JT_j) via a public channel lead to establish the session key between themselves. Due to the encapsulation of the pseudo random number, these digital signatures vis-a-vis the identities of both C_i and NS_j (or JT_j) are used to be dynamic and it will change in every sessions. Thus, the proposed scheme provides user and Big Data service server anonymity.

Proposition 9: HEAP assists untraceability of user and Big Data service servers.

Proof: Suppose an adversary $\mathcal{A}dv$ eavesdrops the message set $\{MC_7, MC_8\}$ and tries to extract the original identities of C_i and NS_j (or JT_j). In this connection, $\mathcal{A}dv$ extracts $\mathcal{MTU} = h(C_{ID}||r_i\ ||HCA_{ID}||\ R_{C_i}^{CMS})$, $TNS_{ID} = h(NS_{ID}\ ||HSA_{ID}||\ rss_1)$ (or $TJT_{ID} = h(JT_{ID}\ ||HSA_{ID}||\ rss_2)$), $V_{cc} = R_C \cdot \mathbb{H}_2(\mathbb{H}_1\ (\mathcal{MTU}\ ||OTK_{CMS,C})) + S_C \cdot \mathbb{H}_2(\psi_{C_i}^1)$ (mod q) and $V_{ns} = R_{NS} \cdot \mathbb{H}_2(\mathbb{H}_1(TNS_{ID}\ ||OTK_{CMS,C})) + S_{NS} \cdot \mathbb{H}_2(P_{ns})$ (mod q) parameters. Note here, these four parameters are implicitly derived from the original identity of either user C_i or service server NS_j (or JT_j), respectively. The adversary $\mathcal{A}dv$ can not trace the actual identities of C_i and NS_j (or JT_j) due to the adoption of collision-resistant cryptographic one way hash function towards the identity transformation. Thus, the proposed scheme satisfies the untraceability property.

Proposition 10: HEAP is resilient against offline dictionary attacks.

Proof: To make the proposed protocol resilient against offline dictionary attacks, C_i transforms his actual password PWD using a system generated random secret r_i, a server generated random nonce $R_{C_i}^{CMS}$ and two hash functions ($h(\cdot)$) as $PWD^* = h(h(r_i||\ PWD)||\ R_{C_i}^{CMS})$, and later store this masked password (PWD^*) into NMS and ES servers.

Suppose a privileged insider acting as an adversary $\mathcal{A}dv$ compromises the dictionary of passwords from NMS (or ES) and tries to reveal the actual password of C_i incorporating offline password guessing attacks. But, $\mathcal{A}dv$ can not extract C_i's actual password due to lack of knowledge about r_i and $R_{C_i}^{CMS}$ parameters and usage of the collision-resistant cryptographic one way hash function towards password transformation. The same technique has been followed to protect $BDSP$'s original password from the offline password guessing attacks.

Proposition 11: HEAP is robust against ciphertext-only attacks (COA) on C_i's or $BDSP/BA$'s password.

Proof: Suppose during the single sign-on process, a passive adversary $\mathcal{A}dv$ listening the communication channel between C_i and CMS for a particular session say S_t, and eavesdrops the exchanged messages say MC_1, MC_2, MC_3, MC_4. $\mathcal{A}dv$ repeats this process for multiple sessions say $S_t, S_{(t+1)}, S_{(t+2)}, \cdots, S_{(t+n)}$ and collects a set of messages say $\{\overleftarrow{MC_1}, \overleftarrow{MC_2}, \overleftarrow{MC_3}, \overleftarrow{MC_4}\}$. Form these set of messages, $\mathcal{A}$ picks $\overleftarrow{MC_4}$ and extracts $\overleftarrow{Z_4'}$. Note here, $\overleftarrow{Z_4'}$ is a set of ciphertexts that are directly associated with C_i's password. Now, from $\overleftarrow{Z_4'}$ the adversary $\mathcal{A}dv$ tries to find out the actual password of C_i. Since, $\mathcal{A}dv$ does not have any knowledge about $R_{C_i}^{CMS}$ and r_i, so it is hard to guess the password from the ciphertexts set. In the similar way, HEAP is also robust against COA on $BDSP/BA$'s password. Thus, completes the proof.

Proposition 12: HEAP is robust against stolen-verifier attacks.

Proof: Suppose a privileged insider acting as an adversary $\mathcal{A}dv$ steals C_i's masked identity and C_i's transformed password (i.e., $\mathcal{MTU} = h(C_{ID}||r_i\ ||HCA_{ID}||\ R_{C_i}^{CMS})$, $PWD^* = h(h(r_i||\ PWD)||R_{C_i}^{CMS})$ from NMS's (or ES's) database and tries to login into a workstation using HCA_j. In this regard, HCA_j computes $\mathcal{EI}' = h(\mathcal{MTU}||\ PWD^*)$ and search the same into the workstation's cookies. Since, HCA_j does not find such an entry into the cookies, it is obvious that HCA_j rejects $\mathcal{A}dv$'s request. In order to satisfy the aforesaid search condition successfully, $\mathcal{A}dv$ needs the knowledge about the original user identity (C_{ID}) and password (PWD) of C_i instead the masked identity and masked password of C_i. In the same way, the proposed scheme does not allow an adversary $\mathcal{A}dv$ to login into the system by stealing $BDSP/BA$'s credentials from the server (NMS or ES). Hence, we can conclude that, the proposed scheme protects stolen-verifier attacks.

Proposition 13: HEAP is resilient against impersonation attacks.

Proof: In order to access the Big Data storage and processing services from the remote service server (NS_j or JT_j), an adversary $\mathcal{A}dv$ initially requires the actual identity and password of C_i for sign in into the local workstation. In a similar way, to enrol the service servers of a Hadoop cluster, $\mathcal{A}dv$ needs the original identity and password of $BDSP$. Since, $\mathcal{A}dv$ does not have these parameters so it is computationally intractable to make a valid sign in request through HCA_j or HSA_k.

Further, during session key establishment, $\mathcal{A}dv$ does not have any means to steal the C_i's (or NS_j's) information to achieve mutual authentication in the presence of CMS. Because, retrieval of $OTK_{CMS,C}$ from C_{token} is computationally hard. Moreover, during this mutual authentication task C_i verifies both CMS and NS_j whereas NS_j checks the legitimacy of both CMS and C_i seperately before establishment of the session key between themselves using the proposed digital signature based verification strategy. Since, the adversary $\mathcal{A}dv$ does not have $OTK_{CMS,C}$, $OTK_{CMS,NS}$, SID_{NS}^j, S_C, S_{NS}, R_C and R_{NS} parameters he would not be able to generate a valid digital signature. Thus, we remark that the proposed protocol has ability to protect user and service server impersonation attacks.

Proposition 14: Heap protects replay attacks.

Proof: To resist replay attacks during mutual authentication and single sign-on task, CMS can keep $\psi_{C_i}^1$ and $\psi_{C_i}^2$ in its system cache memory temporarily. CMS initially verifies if $\psi_{C_i}^{1*} = \psi_{C_i}^1$ and $\psi_{C_i}^{2*} = \psi_{C_i}^2$. If both of these conditions are valid then C_i's request message is treated as replay message, otherwise; CMS updates $\psi_{C_i}^1$ and $\psi_{C_i}^2$ with $\psi_{C_i}^{1*}$ and $\psi_{C_i}^{2*}$ in its system cache memory. Further, in order to

protect the replay attacks during mutual authentication and cluster registration (service servers registration) phase, *NMS* can store $\lambda^1_{C_i}$ and $\lambda^2_{C_i}$ into its cache memory temporarily. In a similar way, we can alleviate the replay attacks during session key formation task between C_i and NS_j (or JT_j) (also see Remark 3) and the other phases of the proposed protocol.

Proposition 15: HEAP is resilient against server spoofing attacks.

Proof: To impersonate both the server i.e., *CMS* and *NMS* to the end user C_i, an adversary $\mathcal{A}dv$ needs to generate the valid $V_{cnew} = \pi^2_{CMS} + S_{NMS} \cdot \mathbb{H}_2(\mathbb{H}_1(K_{NMS,C}))$ (mod q) for the message MC_4 to satisfy the verification condition at user-side (C_i) as $Temp^1_C = Temp^2_C$. It is obvious that the adversary $\mathcal{A}dv$ cannot achieved without having the knowledge of S_{CMS}, S_{NMS} and $K_{C,CMS}$ parameters. In a similar way, it is computationally intractable to impersonate both the servers i.e., *NMS* and *CMS* to the Big Data service provider *BDSP*. Further, to impersonate the server NS_j to C_i, the adversary $\mathcal{A}dv$ needs the knowledge about S_{NS}, $OTK_{CMS,NS}$ and SID^j_{NS} parameters to generate the valid V_{ns}. Thus, we can remark that HEAP can resists server spoofing attacks.

Proposition 16: HEAP protects server-side Single point Of Failure (SOF) and Single point Of Vulnerability (SOV) issues.

Proof: According to the proposed HEAP-KDC architecture, *CMS* interfaces with the clients whereas *NMS* interacts with the service providers and *ES* is reachable offline to both client and service provider at the time of principal registration process. Under the basic assumption of HEAP as discussed in Section V-C, the *CMS* server: which is the front server to the client (C_i): keeps the transformed identities ($\mathcal{MTU}$s) of clients, the dictionary of $R^{CMS}_{C_i}$, the secert credentials (SID^j_{NS}, SID^j_{JT}, $K_{CMS,BDSP}$ and $BDSP^*_{PWD}$) of service providers (*BDSP*s) and service servers (NS_j or JT_js), wherein the *NMS* server: which is the front-end server to the service providers (*BDSP*s): stores the client (C_i) secret credentials (PWD^*s, $K_{NMS,C}$), the dictionary of R^{NMS}_{BDSP} and the transformed identities ($\mathcal{MTU}'$s) of service providers. In addition, *ES* keeps all the credentials of *CMS* and *NMS* servers in its custody for future use. In such a settings, two servers (*CMS* and *NMS*) are actively involved (two-server based handshaking) to achieve authentication of C_i (or *BDSP*) at the time of single-sign on (or service server registration) process. Since, the secret credentials of C_i (or *BDSP*) is distributed into two different servers, therefore it is resilient against SOV issue. Further, from Remark 4, we can observe that the proposed authentication framework resists SOF issue. Thus, completes the proof.

Proposition 17: HEAP provides both forward and backward secrecy.

Proof: To measure the forward and backward secrecy of the proposed protocol, we consider the simultaneous leakage of C_i's primary secret namely password *PWD* in the form of C_{token} and its impact on the past and future session key security.

Suppose in a particular session, to establish a session key between C_i and NS_j (or JT_j), a pseudo random number say R_C is chosen by C_i whereas another pseudo random number say R_{NS} (or R_{JT}) is selected by NS_j. Although, both $\psi^1_{C_i} = R_C \cdot \mathcal{G}$ and $P_{ns} = R_{NS} \cdot \mathcal{G}$ are exchanged between C_i and NS_j via a public channel, it is computationally hard to reveal R_C or R_{NS} (or R_{JT}) from $\psi^1_{C_i}$ or P_{ns} (or P_{jt}), respectively due to the intractability of ECDLP (see Section IV). Further, it is also impossible to compute $R_C \cdot R_{NS} \cdot \mathcal{G}$ or $R_{NS} \cdot R_C \cdot \mathcal{G}$ after getting $P_{ns} = R_{NS} \cdot \mathcal{G}$ or $\psi^1_{C_i} = R_C \cdot \mathcal{G}$ via a public channel due to the intractability of ECDDHP (refer Section IV). However, the session key $SK_{C,NS} = h(R_C \cdot P_{ns} || \ OTK_{CMS,C}) = h(R_{NS} \cdot \psi^1_{C_i} || \ OTK_{CMS,C})$ between C_i and NS_j has derived from R_C, R_{NS}, $\psi^1_{C_i}$, P_{ns} and $OTK_{CMS,C}$ parameters and C_i's password *PWD* has nothing to do with this computation. Thus, our proposed protocol achieves both forward and backward secrecy.

Proposition 18: HEAP provides mutual authentication.

Proof: During single sign-on process, C_i checks the legitimacy of both *CMS* and *NMS*, and vice versa by verifying the following three conditions: (i) $\mathcal{MTU}^* = \mathcal{MTU}$ && $CMS'_{ID} = CMS_{ID}$, (ii) $Temp^1_C = Temp^2_C$ and (iii) $Temp^1_{CMS} = Temp^2_{CMS}$ as discussed in Step DKA6, Step DKA8 and Step DKA9, respectively (refer Section V-D.6). Further, at the time of session key establishment between C_i and NS_j, both of them verify their legitimacy along with *CMS*'s legitimacy utilizing the following two conditions: (a) $\mathcal{MTU}' = \mathcal{MTU}$ && $Temp^{NS}_1 = Temp^{NS}_2$ and (b) $Temp^{C'}_1 = Temp^{C'}_2$ as elaborated in Step SKABSSA4 and Step SKABSSA7, respectively (see Section V-D.8. Thus, we can say that the proposed protocol achieves mutual authentication.

Proposition 19: HEAP provides dependable authentication services.

Proof: From Proposition 16, we can observe that the proposed authentication framework resolves the SOF and SOV issues of the key distribution center (HEAP-KDC). In such a setting, the failure or compromise of a server (*CMS* or *NMS* or *ES*) can not increase the downtime of the authentication system at mission critical situations.

According to the proposed authentication architecture, *CMS* has all the security credentials related to service providers, Hadoop cluster vis-a-vis service servers information and masked identities of users. *NMS* has all the security credentials related to C_i, service providers masked identities and service servers masked identities. *ES* is having all the secret credentials of each principals which is the union of both *CMS* and *NMS*. Now, say for instance, if the active *CMS* suddenly fails and all its security credentials are lost due to some hardware related issues then a stand-by *CMS* will be restored the whole system by configuring it from the back-up server say *ES* via a secure channel. In a similar way, the failure of an active *NMS* can be restored by a stand-by *NMS*. Further, if the trusted server say *ES* fails then the maintenance engineer and system administrator easily rebuild it from both active *CMS* and active *NMS* servers. Note here, a trusted server *ES* (operates in an offline mode)

is currently having all the secret credentials related to each principal of the system including the master *CMS* and *NMS*. It may also be noted that, we consider here only one server (*CMS* or *NMS* or *ES*) can fail at a particular point of time. Therefore, from the above discussion, we can remark that both maintenance engineer and system administrator operates under HEAP-KDC would be able to restore the whole KDC in no time. Thus, the proposed protocol provides dependable authentication services.

Proposition 20: HEAP provides single sign-on facility.

Proof: According to the proposed protocol policy, C_i can access any service servers (that belongs to a particular Hadoop cluster) after authenticating himself utilizing both *CMS* and *NMS* servers (two-server based authentication). This authentication process is a one time task. After that, C_i can make any numbers of service server request from *CMS* throughout the session.

VII. FORMAL SECURITY ANALYSIS USING AVISPA

In HEAP, we have two mutual authentication and session key agreement tasks: 1) to register the service servers vis-a-vis the Hadoop cluster with HEAP-KDC online, service provider's administrator (*BA*) needs to authenticate himself to HEAP-KDC (utilizing both *NMS* and *CMS* servers) and vice versa, and 2) to access the service server NS_j or JT_j from a distinct cluster, the end user C_i needs to authenticate himself utilizing both *CMS* and *NMS* servers and vice versa. To achieve this two cases, we proposed two different authentication strategies as discussed above. In order to validate these two proposed authentication protocols, we utilize a well-known and widely used Internet security protocol verification tool, called AVISPA (Automated Validation of Internet Security Protocols and Applications) [79]–[81]. This tool is used to test whether a security protocol is safe against an active or passive adversary, such as man-in-the-middle and replay attacks.

Currently, AVISPA tool version 1.1[12] is equipped with four implicit back-end model checkers, namely i) On-the-fly Model-Checker (OFMC), ii) Constraint Logic based Attack Searcher (CL-AtSe), iii) SAT-based Model-Checker (SATMC) and iv) Tree Automata based on Automatic Approximations for the Analysis of Security Protocols (TA4SP). Further, each model checker is also equipped with different state-of-the-art automatic analysis algorithms. The internal hierarchy of AVISPA tool and its modules are shown in Figure 25. The following steps are needed to simulate a security protocol in this tool:

Step 1: The proposed protocol needs to be codified into HLPSL (High Level Protocols Specification Language) [81], where HLPSL is the de facto language according to the specification of AVISPA.

Step 2: Save the designed code into a file with hlpsl extension. For example, we save two of our proposed codes (two authentication strategies) into two distinct files as Heap_BNMSfinal.hlpsl and SK_CNS.hlpsl.

Step 3: After creating the file, we only need to execute the command called *"avispa <space> <Filename.hlpsl> <space> −− <model checker name>"*. For example, in our case we execute the following commands in Ubuntu 14.04 LTS platform:

1) *durbadal@durbadal-rec: /Desktop/HEAP_CODE $ avispa Heap_BNMSfinal.hlpsl −− satmc*
2) *durbadal@durbadal-rec: /Desktop/HEAP_CODE $ avispa SK_CNS.hlpsl −− ofmc*

Step 4: During the execution of the above command say *"avispa <space> <Filename.hlpsl> <space> −− <model checker name>"*, the AVISPA tool implicitly translate the "XYZ.hlpsl" file into another file format called Intermediate Format (IF) using the HLPSL2IF translator (see Figure 25).

Step 5: Eventually, the IF file is given to each model checker and the model checker test the proposed protocol is safe or unsafe or inconclusive. The IF file is the required input file format of each aforementioned model checker to test the designed protocol.

We have implemented the codes for two proposed protocols in HLPSL and save them into two different files namely Heap_BNMSfinal.hlpsl and SK_CNS.hlpsl. The detailed description on HLPSL and various protocols implementations in AVISPA are available in [80] and [81]. Under Heap_BNMSfinal.hlpsl file, initially we specify basic roles of all the participants (*BDSP*, *NMS* and *CMS*) and then make composite role for representing different cases or scenarios derived from the basic roles. Similarly, under SK_CNS.hlpsl, we present the basic roles for C_i, *CMS*, *NMS* and NS_j and construct the composite role (or session) involving all the participants.

We simulate both the files using AVISPA under the widely-used OFMC and SATMC back-ends and summarized the simulation results in Figure 26 and 27. The simulation results show that the proposed two protocols in HEAP are safe from man-in-the-middle and replay attacks.

VIII. PERFORMANCE ANALYSIS

This section analyzes the performance of HEAP. Currently, HEAP protocol consists of two modules: (1) Big Data service provider module and (2) client module. We evaluate the performance of HEAP based on these module and present them in Table 4 and Table 5. During the performance analysis, we consider four different metrics as follows:

1) *CPU usage or Computation Time (CT) in terms of seconds* – It tells about the execution time or CPU usage (in terms of seconds) of different cryptographic operations that we have used throughout the proposed protocol. For example, a cryptographic one-way hash function (SHA-1), modular exponentiation, ECC scalar multiplication, AES-128 bits stateless CBC mode of encryption or decryption. An approximate CT for the

[12] AVISPA Project: http://www.avispa-project.org/

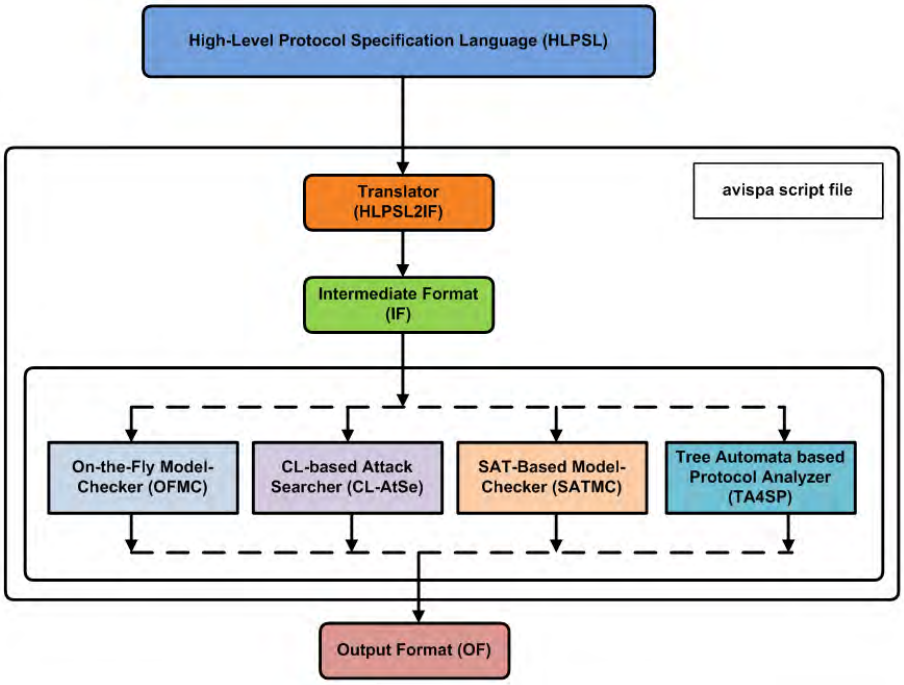

FIGURE 25. Building blocks of AVISPA (Source: [79]).

```
% OFMC
% Version of 2006/02/13
SUMMARY
  SAFE
DETAILS
  BOUNDED_NUMBER_OF_SESSIONS
PROTOCOL
  /opt/avispa-1.1/testsuite
  /results/Heap_BNMSfinal.if
GOAL
  as_specified
BACKEND
  OFMC
COMMENTS
STATISTICS
  parseTime: 0.00s
  searchTime: 2.41s
  visitedNodes: 859 nodes
  depth: 10 plies
```

```
% OFMC
% Version of 2006/02/13
SUMMARY
  SAFE
DETAILS
  BOUNDED_NUMBER_OF_SES
PROTOCOL
  /opt/avispa-1.1/testsuite
  /results/SK_CNS.if
GOAL
  as_specified
BACKEND
  OFMC
COMMENTS
STATISTICS
  parseTime: 0.00s
  searchTime: 22.17s
  visitedNodes: 3637 nodes
  depth: 12 plies
```

FIGURE 26. Analysis of results under the OFMC backend.

aforesaid operations are taken form [63] and it is summarized in Table 3. In the proposed protocol phases, we use logical *XOR* operation but it is not depicted in Table 3. Because, this operation takes a negligible amount of time say, $T_{xor} = 10^{-9}$ seconds as compare to the other as mentioned in Table 3. Therefore, we do not consider T_{xor} for CT computation for both service provider and client modules (refer Table 4 and Table 5).

2) *Communication Overhead (CO) in terms of bits* – Suppose an entity say C_i sends a message M_i to another entity say CMS and $|M_i|$ represents the bit length of the message M_i by summing up the bit length of its individual component. More precisely, if $M_i = \{c_1, \cdots, c_n\}$ then $|M_i| = \{|c_1| + \cdots + |c_n|\}$ or $|M_i| = \sum_{j=1}^{n}(c_j)$. For example, in single sign-on and dynamic key establishment phase, C_i sends $MC_1 = \{\mathcal{MTU}, CMS_{ID}, \psi^1_{C_i}, \psi^2_{C_i}, Cert_C\}$. Here, $|\mathcal{MTU}| = 160$ bits, $|CMS_{ID}| = 32$ bits, $|\psi^1_{C_i}| = 160$ bits, $|\psi^2_{C_i}| = 160$ bits and $|Cert_C| = 160$ bits. So, $|MC_1| = (160 + 32 + 160 + 160 + 160) = 672$ bits. Therefore, the communication overhead of message MC_1 is 672 bits. Similarly, we calculate the CO for all the other messages and are depicted in Table 4 and Table 5. Note here, for ease of CO calculation, we assume here: (i) the bit length of each certificate is equal to the bit length of the public-key (i.e., 160 bits ECC key), (ii) the bit length of the masked identity is 160 bits, (iii) $|nonce| = 32$ bits, (iv) the bit length of the server's original identity is 32 bits and (v) the bit length of cipher-text or plain-text block is 128 bits (using AES-128 bit encryption/decryption policy).

3) *Storage Cost (SC) in terms of bits* – To achieve a particular task (namely user registration, user login, etc.) in the proposed protocol, few security credentials are need to be stored previously either in client-side

TABLE 3. Rough estimation of computation time reported in [63].

Operation	Notations	Approximate computation cost (in seconds)
Hash (SHA-1)	T_{hash}	0.00032
Encryption or Decryption	$T_E/T_D\ (= T_S)$	0.0056
Modular exponentiation	T_{mod}	0.0192
ECC scalar multiplication	T_{ecsm}	0.0171

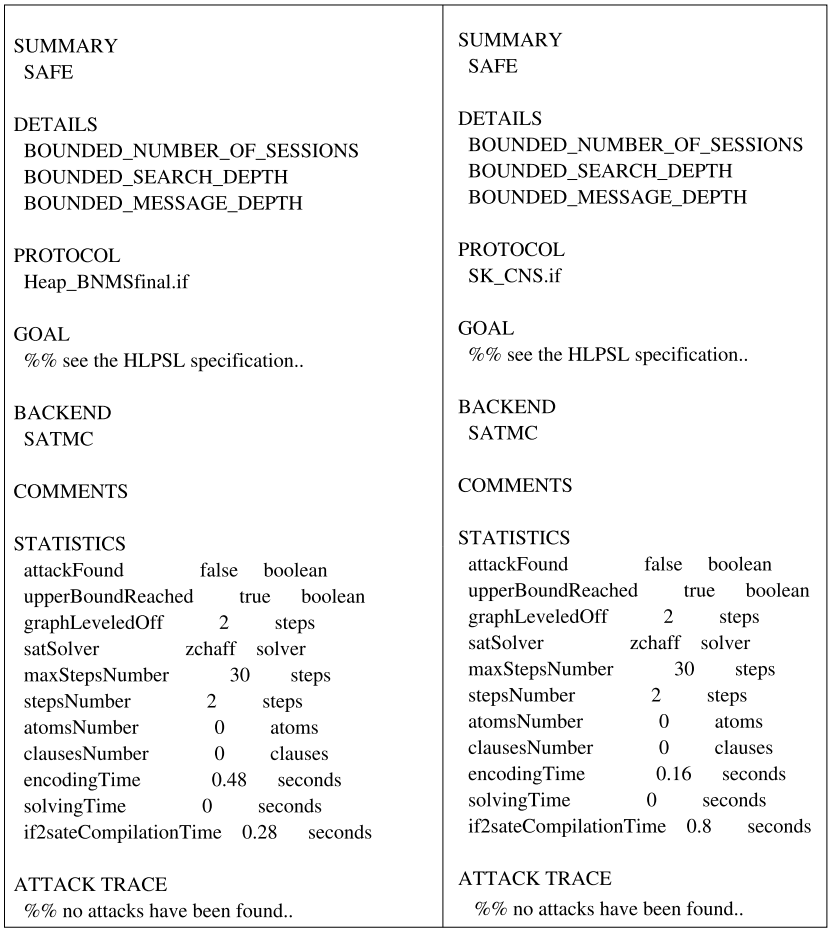

FIGURE 27. Analysis of results under the SATMC backend.

or server-side or both. The storage cost tells about this pre-loaded credentials in terms of bits. For example, to achieve user login, HCA_i needs to load r_i^* and $USPW_i$ (total $|r_i^*| + |USPW| = 320$ bits) credentials into C_i's workstation. Similarly, to accomplish the service provider login, HSA_j needs to load d^* and $BDPW$ (total $|d^*| + |BDPW| = 320$ bits) into $BDSP$'s workstation. We highlights the SC for each phase of the proposed protocol in Table 4 and Table 5. Note here, to achieve a successful user registration and single sign-on task, NMS needs to keep $(|K_{NMS,C}| + |D_{C_{ID}}| + |D_{PWD}|)$, a total of $(64+64+64)$ bits $= 192$ bits, and both NMS and CMS need to store $(|\mathcal{MTU}| + |PWD^*| + |R_{C_i}^{CMS}| + |K_{NMS,C}|)$, a total of $(160+160+32+64)$ bits $= 416$ bits, respectively. Further, to make a Big Data storage or processing service server ticket request, the proposed protocol needs to store the masked identity of either NS_j or JT_j (160 bits) into HCA_i.

4) *Communication Rounds (CR)* – A single communication round represents a one-way message transmission. In this regards, we compute CR for each phases of the proposed protocol and it is shown in Table 4 and Table 5.

Tables 4 and 5 show that the proposed protocol is quite efficient in terms of communication, storage and computation cost considering all the participants. Consider C_i with HCA_i, for instance; it needs 0.00128 seconds and 0.17588 seconds, a total of 0.17716 seconds $\approx$ 177 milliseconds (ms) (refer $Scenario_2$ from Table 5) to login into the system and validate the legitimacy of HEAP-KDC, respectively. After C_i's single sign-on (or establishment of the session key with CMS), the C_i needs approximately 0.10072 seconds $(1T_E + 1T_D + 4T_{ecsm} + 1T_{mod} + 6T_{hash}) \approx 101$ ms to access a particular service server (i.e., either NS_j or JT_j) from any Hadoop cluster (which is registered with the HEAP-KDC) followed by a mutual authentication and session key agreement phase. According to the computational cost analysis of the proposed protocol, service provider enrollment and password updation tasks will take $\approx$ 188 ms and $\approx$ 18.08 ms and with the same cost, user achieves his registration and password change phases. In addition to this, a service provider can seamlessly enroll his own Hadoop cluster with the HEAP-KDC by compromising only 33.6 ms (here, we assume that the Hadoop cluster consists of a single storage service server and a single processing server). In spite of this, we compare our scheme with the existing state of the art authentication protocols as follows.

TABLE 4. Performance of HEAP considering only Big Data service provider module.

Protocol phases	Performance metric	Participants ($BDSP$ + NMS + CMS + ES)	Overall costs per phases (CO, CR, SC, CT)
Service provider registration	Communication overhead	(128+128+32)+(768)+(384+160+32+32)+(32+32+160) bits [++]	(1888 bits, 6, 192 bits, 0.1877 sec)
	Communication rounds	6[++]	
	Storage cost	192 bits	
	Computation cost	$12T_{ecsm}/(5T_{ecsm}) + 2T_{mod} + 3T_E + 2T_D + 3T_{xor} + 9T_{hash}/(4T_{hash})$	
Service provider login	Communication overhead	None	(0 bit, 0, 320 bits, 0.00128 sec)
	Communication rounds	None	
	Storage cost	320 bits	
	Computation cost	$3T_{xor} + 4T_{hash}$	
Authenticated key agreement	Communication overhead	$Scenario_1$: $\|M_1\|+\cdots+\|M_4\|$= 3200 bits $Scenario_2$: $\|M_1\|+\cdots+\|M_4\|$= 2560 bits	(3200 [or 2560] bits, 4, 416 bits, 0.21008 [or 0.17588] sec)
	Communication rounds	4	
	Storage cost	160+160+64+32 bits = 416 bits	
	Computation cost	$Case_1 : 15T_{ecsm}/(7T_{ecsm}) + 3T_{mod}/(1T_{mod}) + 3T_E + 3T_D + 15T_{hash}/(11T_{hash})$ $Case_2 : 15T_{ecsm}/(9T_{ecsm}) + 3T_{mod}/(1T_{Mod}) + 3T_E + 3T_D + 15T_{hash}/(11T_{hash})$	
Service server enrollment	Communication overhead	160+32+32+(896+256)+160+32+32 bits	(1600 bits, 2, 0 bit, 0.0336 sec)
	Communication rounds	2	
	Storage cost	None	
	Computation cost	$2T_E + 4T_D$	
Password updation	Communication overhead	32+160+32+384+32+32+160 bits [++]	(832 bits, 2, 320 bits, 0.01808 sec)
	Communication rounds	2[++]	
	Storage cost	320 bits	
	Computation cost	$1T_E + 2T_D + 4T_{hash}$	

Note: $X/(Y)$ – It signifies that an entity does X computation in real time and Y computation in offline mode; $Scenario_1$ – Dynamic public-private key-pair and dynamic certificate usage for each principal (except ES) in each session; $Scenario_2$ – Static public-private key-pair and certificate usage for a long period of time; $Case_1$ – first time login and session key establishment utilizing $Scenario_1$; $Case_2$ – next consecutive login and session key establishment incorporating $Scenario_2$; [++]For ease of calculation, we ignore internal broadcast messages among NMS, CMS and ES, respectively.

TABLE 5. Performance of HEAP considering only Client module.

Protocol phases	Performance metric	Participants (C_i + NMS + CMS + ES)	Overall costs per phases (CO, CR, SC, CT)
User registration	Communication overhead	(128+128+32)+(768)+(384+160+32+32)+(32+32+160) bits [**]	(1888 bits, 6, 192 bits, 0.1877 sec)
	Communication rounds	6[**]	
	Storage cost	192 bits	
	Computation cost	$12T_{ecsm}/(5T_{ecsm}) + 2T_{mod} + 3T_E + 2T_D + 3T_{xor}$ $+9T_{hash}/(4T_{hash})$	
User login	Communication overhead	None	(0 bits, 0, 320 bits, 0.00128 sec)
	Communication rounds	None	
	Storage cost	320 bits	
	Computation cost	$3T_{xor} + 4T_{hash}$	
Mutual Authentication and Single sign-on	Communication overhead	$Scenario_1 : MC_1 + \cdots + MC_4$= 3520 bits $Scenario_2 : MC_1 + \cdots + MC_4$ = 2880 bits	(3520 [or 2880] bits, 4, 416 bits, 0.21008 [or 0.17588] sec)
	Communication rounds	4	
	Storage cost	160+160+64+32 bits	
	Computation cost	$Case_1 : 15T_{ecsm}/(7T_{ecsm}) + 3T_{mod}/(1T_{mod}) + 3T_E + 3T_D$ $+15T_{hash}/(11T_{hash})$ $Case_2 : 15T_{ecsm}/(9T_{ecsm}) + 3T_{mod}/(1T_{Mod}) + 3T_E$ $+3T_D + 15T_{hash}/(11T_{hash})$	
Service server ticket granting	Communication overhead	640+384+384+384 bits	(1792 bits, 2, 0 bits, 0.01712 sec)
	Communication rounds	2	
	Storage cost	None	
	Computation cost	$2T_E + 1T_D + 1T_{hash}$	
Session key agreement with service server	Communication overhead	160 + 160 + 160 + 160 + 384 + 160 + 800 bits	(1984 bits, 2, 160 bits, 0.17936 sec)
	Communication rounds	2	
	Storage cost	160 bits	
	Computation cost	$5T_{ecsm}/(1T_{ecsm}) + 1T_{mod} + 7T_{hash}/(1T_{hash})+$ $6T_{ecsm}/(2T_{ecsm}) + 1T_{mod} + 8T_{hash}/(1T_{hash})$	
Password updation	Communication overhead	32+160+32+384+32+32+160 bits	(832 bits, 2, 320, 0.01808 sec)
	Communication rounds	2 [**]	
	Storage cost	320 bits	
	Computation cost	$1T_E + 2T_D + 4T_{hash}$	

Note: $X/(Y)$ – It signifies that an entity does X computation in real time and Y computation in offline mode; $Scenario_1$ – Dynamic public-private key-pair and dynamic certificate usage for each principal (except ES) in each session; $Scenario_2$ – Static public-private key-pair and certificate usage for a long period of time; $Case_1$ – first time login and session key establishment utilizing $Scenario_1$; $Case_2$ – next consecutive login and session key establishment incorporating $Scenario_2$; [**]For ease of calculation, we ignore internal broadcast messages among NMS, CMS and ES, respectively.

A. COMPARATIVE ANALYSIS

This section compares the proposed protocol HEAP with other state of the art authentication strategies. For comparison, we consider only login and authentication phases of the proposed protocol. Further, we consider three crucial performance metrics namely communication

TABLE 6. Summary of computation cost analysis.

State-of-the-art authentication schemes	Computation cost (in mathematical notations)	A rough estimation of CPU utilization (in seconds)
Karla and Sood [46]	$9T_{hash} + 7T_{escm}$	0.12258
S. Kumari *et al.* [47]	$7T_{hash} + 8T_{escm}$	0.13904
Odelu *et al.* [60]	$25T_{hash} + 6T_S + 1T_F + 6T_{ecsm}$	0.1613
S. Kumari *et al.* [62]	$16T_{hash} + 8T_{ecsm} + 2T_{BH}$	0.17612
Shen *et al.* [57]	$17T_{hash} + 6T_{ecsm}$	0.10804
Yoon and Yoo [55]	$16T_{hash} + 4T_{ecsm}$	0.07352
Mishra *et al.* [56]	$1T_{BH} + 16T_{hash}$	0.02222
Wu *et al.* [58]	$13T_{hash} + 2T_{ecsm} + 4T_S + 1T_F$	0.07786
He and Wang [59]	$21T_{hash} + 8T_{escm}$	0.14352
Wazid *et al.* [61]	$22T_{hash} + 4T_S + T_F$	0.04654
Jangirala *et al.* [63]	$31T_{hash} + 4T_{CM} + T_F$	0.09542
Katz *et al.* [66]	$21T_{mod} + \epsilon$	0.4032
Yi *et al.* [68]	$32T_{mod} + \epsilon$	0.6144
JWX Protocol [65]	$18T_{mod} + \epsilon$	0.3456
Yang *et al.* [64]	$11T_{mod}/(4T_{mod}) + \epsilon$	0.1728
Yi *et al.* [67]	$14T_{mod} + \epsilon$	0.2688
Jangirala *et al.* [17]	$12T_{hash} + 8T_{ecsm} + 6T_{epa}$	0.27744
Proposed scheme	$15T_{ecsm}/(9T_{ecsm}) + 3T_{mod}/(1T_{Mod})$ $+6T_S + 19T_{hash}/(11T_{hash})$	0.17716

Note: Here, we consider $T_F = T_{epa} = T_{BH} = T_{CM} = T_{escm} \approx 0.0171$ seconds; ϵ represents some other costs (exponentiation and hash operations cost) that are involved in the particular scheme, but for ease of calculation, in this work, we consider average computation cost.

TABLE 7. Summary of communication overhead analysis.

State-of-the-art authentication schemes	Communication cost (in bits)
Karla and Sood [46]	1280
S. Kumari *et al.* [47]	1760
Odelu *et al.* [60]	2944
S. Kumari *et al.* [62]	2880
Shen *et al.* [57]	1856
Yoon and Yoo [55]	2496
Mishra *et al.* [56]	1152
Wu *et al.* [58]	1856
He and Wang [59]	3520
Wazid *et al.* [61]	3232
Jangirala *et al.* [63]	1856
Katz *et al.* [66]	4480
Yi *et al.* [68]	3840
JWX Protocol [65]	4480
Yang *et al.* [64]	3520
Yi *et al.* [67]	4000
Jangirala *et al.* [17]	2496
Proposed scheme	2880

TABLE 8. Summary of storage cost analysis.

State-of-the-art authentication schemes	Storage cost (in bits)
Karla and Sood [46]	576
S. Kumari *et al.* [47]	480
Odelu *et al.* [60]	N/A
S. Kumari *et al.* [62]	N/A
Shen *et al.* [57]	N/A
Yoon and Yoo [55]	N/A
Mishra *et al.* [56]	N/A
Wu *et al.* [58]	N/A
He and Wang [59]	N/A
Wazid *et al.* [61]	N/A
Jangirala *et al.* [63]	N/A
Katz *et al.* [66]	384
Yi *et al.* [68]	384
JWX Protocol [65]	384
Yang *et al.* [64]	384
Yi *et al.* [67]	384
Jangirala *et al.* [17]	N/A
Proposed scheme	416

Note: Karla and Sood scheme considers 224 bits ECC but Kumari *et al.* and our proposed scheme adopt 160 bits ECC; N/A – the particular scheme does not considers the storage cost during performance analysis.

overhead, CPU usage (or computation time) and storage cost as compared with other schemes. The detail comparative analysis is discussed as follows:

1) COMPARISON OF COMPUTATION COSTS

In order to perform the computational cost analysis, we consider various schemes related to Big Data, Cloud and Internet of Thing (IoT) platforms, and are available in recent literature [17], [46], [47], [55]–[63]. Further, we consider various two-server based PAKE protocols [64]–[68] for the analysis. Since its inception, the schemes [17], [46], [47], [55]–[63] follow single server based authentication strategy whereas others [64]–[68] follow two-server based analogy. From Table 6, we can observe that our proposed scheme is better than that of existing two-server based approaches [64]–[68]. Mean while, the proposed scheme is also quite comparable with the traditional single server based authentication approaches.

2) COMPARISON OF COMMUNICATION OVERHEADS

In communication overhead analysis, we compare the proposed protocol with the aforesaid schemes and summarize it in Table 7. It is easy to say that our scheme is efficient (only 3520 bits need to be transferred between C_i and HEAP-KDC for first time authentication or single-sign on and for future single-sign on only 2880 bits are required) as compare to the two-server based authentication schemes [65]–[68] (needs 4480, 3840, 4480 and 4000 bits). Further, we can observe that our scheme has the same CO compared with Kumari *et al.* [62]. In fact, our scheme is quite admirable in

TABLE 9. Summary of comparison in terms of security and functionality features.

SFFs	[46]	[47]	[60]	[62]	[57]	[55]	[56]	[58]	[59]	[61]	[63]	[66]	[68]	[65]	[64]	[67]	[17]	HEAP
SFF_1	✗	✓	✓	✓	✓	✗	✓	✓	✓	✓	✓	N/A	N/A	N/A	N/A	N/A	✓	✓
SFF_2	✗	✓	✓	✓	✗	✗	✓	✓	✓	✓	✓	N/A	N/A	N/A	N/A	N/A	✓	✓
SFF_3	✗	✓	✓	✓	✓	✓	✓	✗	✓	✓	✓	✓	✓	✓	✓	✓	N/A	✓
SFF_4	✓	✓	✓	✓	✓	✗	✗	✗	✗	✓	✓	N/A	N/A	N/A	N/A	N/A	✓	✓
SFF_5	✓	✓	✓	✓	✓	✓	✓	✓	✗	✓	✓	N/A	N/A	N/A	N/A	N/A	✓	✓
SFF_6	✗	✓	✓	✓	✓	✓	✗	✓	✗	✓	✓	N/A	N/A	N/A	N/A	N/A	✓	✓
SFF_7	✗	✓	✓	✓	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
SFF_8	✗	✓	✓	✓	✓	✓	✗	✓	✓	N/A	N/A	✗	✗	✗	✗	✗	✓	✓
SFF_9	N/A	N/A	✓	✓	✓	✗	✓	✓	✗	N/A	N/A	N/A	N/A	N/A	N/A	N/A	✗	✓
SFF_{10}	✗	✓	✓	✓	✓	✓	✓	✓	✗	✓	✓	✓	✓	✓	✓	✓	N/A	✓
SFF_{11}	✗	✗	✓	✓	✓	✓	✓	✗	✗	✓	✓	✗	✗	✗	✗	✗	N/A	✓
SFF_{12}	✗	N/A	N/A	✓	✓	✓	✗	✓	N/A	N/A	✗	N/A	N/A	✗	N/A	N/A	N/A	✓
SFF_{13}	N/A	✓	✓	N/A	N/A	N/A	N/A	✗	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	✓
SFF_{14}	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	N/A	N/A	N/A	N/A	N/A	✗	✓
SFF_{15}	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✓	✓	✓	✓	✗	✓
SFF_{16}	✗	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✗	✗	✗	✗	✗	✗	✗
SFF_{17}	✓	✓	✓	✗	N/A	✓	N/A	N/A	✓	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	✓
SFF_{18}	✓	✓	✓	✗	N/A	N/A	✓	N/A	✓	✓	✓	N/A	N/A	N/A	N/A	N/A	✓	✓
SFF_{19}	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	✓
SFF_{20}	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	✓
SFF_{21}	✗	✓	✓	N/A	N/A	✗	N/A	N/A	N/A	✓	N/A	N/A	N/A	N/A	N/A	N/A	✓	✓
SFF_{22}	✗	✗	✗	✓	✗	✗	✗	✓	✗	✓	✓	✓	✓	✗	✗	✓	✗	✓
SFF_{23}	✓	✓	✓	✗	✗	✗	✓	✗	✗	✓	✓	✗	✗	✗	✗	✗	✓	✓
SFF_{24}	✗	✗	✓	✗	✓	✓	N/A	✓	✓	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	✓
SFF_{25}	✗	✗	✓	✗	✗	✗	✓	✗	✗	✓	✓	N/A	N/A	N/A	N/A	N/A	✓	✓
SFF_{26}	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	N/A	N/A	✓	✗	✓
SFF_{27}	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✓	N/A	✓	✗	N/A	✓	N/A	✓
SFF_{28}	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓
SFF_{29}	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓

Note: SFF_1 – Resists privileged-insider attacks; SFF_2 – Provides user or device anonymity; SFF_3 – Resists off-line password guessing attacks; SFF_4 – Resists user impersonation attack; SFF_5 – Resists replay attacks; SFF_6 – Resists server impersonation attacks; SFF_7 – Provides secure mutual authentication; SFF_8 – Provides forward secrecy; SFF_9 – Resists known session-specific temporary information attacks; SFF_{10} – Provides session key security; SFF_{11} – Provides freely password changing facility; SFF_{12} – Provides backward secrecy; SFF_{13} – Resists identity compromisation attacks; SFF_{14} – Resists single point of failure issue; SFF_{15} – Resists single point of vulnerability issue; SFF_{16} – Extra hardware cost involved; SFF_{17} – Resists stolen-verifier attacks; SFF_{18} – Resists man-in-the-middle attacks; SFF_{19} – Resists ciphertext-only attacks; SFF_{20} – Resists chosen plaintext attacks; SFF_{21} – Provides service server anonymity; SFF_{22} – Provides formal verification using RoR model; SFF_{23} – Provides AVISPA-based protocol verification; SFF_{24} – Resists server spoofing attacks; SFF_{25} – Resists denial of service attacks; SFF_{26} – Provides online updation of service server's secret credentials; SFF_{27} – Resists online dictionary attacks; SFF_{28} – Provides fault tolerant authentication framework; SFF_{29} – Provides single sign-on facility;✓– SFF is achieved; ✗– SFF is not achieved; N/A – SFF is not analyzed in this scheme.

terms of the CO as compare with the single server based strategy. The proposed scheme's communication overhead is equal to the CO of He and Wang's scheme [59].

3) COMPARISON OF STORAGE COSTS

The memory usage (or pre-deployed secrets storage cost) is involved to smooth execution of the proposed scheme as compare to the aforementioned schemes are shown in Table 8. We can see that, our scheme is efficient in terms of memory usage (client's workstation as well as server-side) than both Karla and Sood and Kumari *et al.* schemes. Although, our scheme is lagging in terms of storage overhead as compare with the existing state of the art two-server based authentication strategies. The remaining schemes (shown in Table 8) are not explicitly analyzed the storage overhead in their works. So, we represent it as N/A in Table 8.

Besides all the above cost factors, the proposed scheme supports several security and functional features (SFFs) as compare to the other schemes. This SFFs are discussed as follows.

4) COMPARISON OF SECURITY AND FUNCTIONAL FEATURES

In this section, we discuss several Security and Functional Features (SFFS) of the proposed protocol and compare it with other existing state of the art schemes. We summarize this discussion in Table 9. Form Table 9, it has been observed that the proposed scheme HEAP fulfills various security and functional features as compare with different state of the art schemes.

IX. DISCUSSIONS

This section summarizes the major potentials of the proposed scheme. In this regard, various appealing features of the proposed user identity and password-assisted two-server authentication and key exchange framework are highlighted as follows:

1) With the proposed enrolment strategy, a Big Data service provider's administrator can securely deploy his cluster vis-a-vis service servers online with authentication service provider (HEAP-KDC). In order to achieve this, the administrator needs to login into the system using his identity and password only. Thus, the cluster enrollment policy is scalable and user friendly in nature.
2) The proposed HEAP-KDC is robust against single point of failure and single point of vulnerability. Further, it could tolerate various well-known attacks. Hence, HEAP can able to provide dependable and secure authentication service 24×7 to the customers.
3) With the proposed two-server based single sign-on scheme, the client can access any number of Namenode servers (NS_js) or JobTrackers (JT_js) from a Hadoop cluster by logging in into the system only once. During single sign-on process, a secure mutual authentication process takes place which ensures the legitimacy of both C_i and the dual-server (*CMS* and *NMS*).
4) The adoptation of two-factor authentication (password and delegated token) and dual-server based session key establishment strategy makes the proposed authentication system more robust, cost effective and user friendly.
5) The key rollover problem is a crucial issue for traditional user registration policy, where during session key establishment, a long-term secret key password) is used to build a secret channel between user and Registration Authority (*RA*). With this settings, it is very difficult for a user to change or update his long-term credential i.e., password into *RA* (a centralized server) for security reasons. Since, the proposed protocol has a provision to update user's password online and has an ability to recover the password utilizing a out-of-band channel (i.e., postal network) or a valid email id (or registered mobile number), it is obvious that the current settings of the proposed approach mitigates the key rollover problem.
6) The utilization of 160 bits public/private key (ECC) and 128 bits symmetric key (AES) as compared to 4096 bits key-size reported in [64]–[68] needs lesser memory space (storage in terms of bits) and reduce communication overheads.
7) The proposed two-server based authentication and key exchange protocol does not have any compatibility hurdle with the traditional single-server based approaches [17], [46], [47], [62], but yields better dependability (fault-tolerant in terms of secret credentials distribution and replication) and security (resists more well-known attacks). The utilization of two separate application instances say *HCA* and *HSA* divide the users domain into two distinct categories, hence maintenance of both client and service provider is easy. Further, HEAP-KDC encloses a less number of verification parameters (i.e., masked identity, pass-phrase and digital signature) inside CT and NST, that decreases the verification cost for C_i and NS_j (or JT_j), respectively
8) The proposed protocol preserves both user and service server (NS or JT) identities from external as well as internal adversaries (malicious insider control HEAP-KDC's server). In such a provision, it is very difficult to trace the on-going activities among end users', HEAP-KDC's servers and service servers'.

X. CONCLUSION

In this paper, we proposed a new fault tolerant two-server authentication and key agreement protocol (HEAP) for Hadoop framework to access secure and privacy preserved Big Data storage and processing services. To achieve this objective, initially, C_i needs to login into its workstation using his identity and password. Then, the single sign-on mechanism vis-a-vis the session key formation task has been carried out in which an end user C_i and *CMS* server establish a session secret key $SK_{C,CMS}$ between themselves with the help of *NMS* server, followed by a mutual authentication process. With this session secret key $SK_{C,CMS}$, C_i can make a service server (Big Data storage or Big Data processing service server) requests to *CMS* server through a secure channel. In the next-level, *CMS* responses C_i with two tokens (C_{token} and NS_{token} or JT_{token}). Utilizing these two tokens, both C_i and NS_j (or JT_j) are mutually authenticate themselves and establish the session key for secure future communication.

In this work, we proposed a new key distribution center (HEAP-KDC), where we can distribute and replicate the security credentials of each principal in such a way that it makes the overall authentication system more dependable and fault-tolerant.

The rigorous security analysis of HEAP under *de facto* ROR simulation (formal analysis) and informal inspection shows that the proposed scheme is provably secure. Moreover, the security of the proposed protocol (HEAP) is also verified utilizing the widely-used AVISPA protocol simulator tool. All these security analysis outputs shows that HEAP is robust against active and passive adversary. In addition, the performance analysis evident that the proposed protocol (HEAP) is effective in terms of computation, communication and storage costs, and comparable with the existing state-of-art schemes.

In the future, we plan to integrate HEAP with a real-world large scale cluster setting (or real-time Hadoop cluster) and try to re-calibrate it further to enhance the security and performance with the real-world deployment.

ACKNOWLEDGMENTS

The authors also thank the anonymous reviewers for their valuable feedback on the paper which helped us to improve its quality and presentation.

REFERENCES

[1] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The Hadoop distributed file system," in *Proc. 26th Symp. Mass Storage Syst. Technol. (MSST)*, May 2010, pp. 1–10.

[2] S. Ghemawat, H. Gobioff, and S. Leung, "The Google file system," *ACM SIGOPS Oper. Syst. Rev.*, vol. 37, no. 5, pp. 29–43, 2003.

[3] O. Rodeh and A. Teperman, "zFS—A scalable distributed file system using object disks," in *Proc. 20th IEEE/11th NASA Goddard Conf. Mass Storage Syst. Technol. (MSST)*, Apr. 2003, pp. 207–218.

[4] S. A. Weil, S. A. Brandt, E. L. Miller, D. D. Long, and C. Maltzahn, "Ceph: A scalable, high-performance distributed file system," in *Proc. 7th Symp. Oper. Syst. Design Implement.*, 2006, pp. 307–320.

[5] O. O'Malley, "Integrating kerberos into apache Hadoop," in *Proc. Kerberos Conf.*, 2010, pp. 26–27.

[6] P. P. Sharma and C. P. Navdeti, "Securing big data Hadoop: A review of security issues, threats and solution," *Int. J. Comput. Sci. Inf. Technol.*, vol. 5, no. 2, pp. 2126–2131, 2014.

[7] S. Jin, S. Yang, X. Zhu, and H. Yin, "Design of a trusted file system based on Hadoop," in *Proc. Int. Conf. Trustworthy Comput. Services*, 2012, pp. 673–680.

[8] H. Zhou and Q. Wen, "A new solution of data security accessing for Hadoop based on CP-ABE," in *Proc. 5th Int. Conf. Softw. Eng. Service Sci. (ICSESS)*, Jun. 2014, pp. 525–528.

[9] F. A. H. Jing, S. B. L. Renfa, and T. C. T. Zhuo, "The research of the data security for cloud disk based on the Hadoop framework," in *Proc. 4th Int. Conf. Intell. Control Inf. Process. (ICICIP)*, Jun. 2013, pp. 293–298.

[10] D. Chattaraj, M. Sarma, and A. K. Das, "A new two-server authentication and key agreement protocol for accessing secure cloud services," *Comput. Netw.*, vol. 131, pp. 144–164, Feb. 2018.

[11] G. S. Sadasivam, K. A. Kumari, and S. Rubika, "A novel authentication service for Hadoop in cloud environment," in *Proc. Int. Conf. Cloud Comput. Emerg. Markets (CCEM)*, 2012, pp. 1–6.

[12] P. Rahul and T. GireeshKumar, "A novel authentication framework for Hadoop," in *Proc. Artif. Intell. Evol. Algorithms Eng. Syst.*, 2015, pp. 333–340.

[13] M. Sarvabhatla, M. C. M. Reddy, and C. S. Vorugunti, "A secure and light weight authentication service in Hadoop using one time pad," *Procedia Comput. Sci.*, vol. 50, pp. 81–86, Mar. 2015.

[14] N. Somu, A. Gangaa, and V. S. S. Sriram, "Authentication service in Hadoop using one time pad," *Indian J. Sci. Technol.*, vol. 7, pp. 56–62, Apr. 2014.

[15] Z. Shen, L. Li, F. Yan, and X. Wu, "Cloud computing system based on trusted computing platform," in *Proc. Int. Conf. Intell. Comput. Technol. Automat. (ICICTA)*, vol. 1, 2010, pp. 942–945.

[16] G. S. Aujla, R. Chaudhary, N. Kumar, A. K. Das, and J. J. P. C. Rodrigues, "SecSVA: Secure storage, verification, and auditing of big data in the cloud environment," *IEEE Commun. Mag.*, vol. 56, no. 1, pp. 78–85, Jan. 2018.

[17] J. Srinivas, A. K. Das, and J. J. P. C. Rodrigues, "2PBDC: Privacy-preserving bigdata collection in cloud environment," *J. Supercomput.*, pp. 1–30, Sep. 2018, doi: 10.1007/s11227-018-2605-1.

[18] A. K. Das, "Analysis and improvement on an efficient biometric-based remote user authentication scheme using smart cards," *IET Inf. Secur.*, vol. 5, no. 3, pp. 145–151, Sep. 2011.

[19] S. Chatterjee, S. Roy, A. K. Das, S. Chattopadhyay, N. Kumar, and A. V. Vasilakos, "Secure biometric-based authentication scheme using chebyshev chaotic map for multi-server environment," *IEEE Trans. Dependable Secure Comput.*, vol. 15, no. 5, pp. 824–839, Sep./Oct. 2018.

[20] M. Wazid, A. K. Das, S. Kumari, X. Li, and F. Wu, "Provably secure biometric-based user authentication and key agreement scheme in cloud computing," *Secur. Commun. Netw.*, vol. 9, no. 17, pp. 4103–4119, 2016.

[21] V. Odelu, A. K. Das, S. Kumari, X. Huang, and M. Wazid, "Provably secure authenticated key agreement scheme for distributed mobile cloud computing services," *Future Gener. Comput. Syst.*, vol. 68, pp. 74–88, Mar. 2017.

[22] V. U. Srinivasan, R. Angal, and A. Sondhi, "OAuth framework," U.S. Patent 8 935 757, Jan. 13, 2015.

[23] L. Kang and X. Zhang, "Identity-based authentication in cloud storage sharing," in *Proc. Int. Conf. Multimedia Inf. Netw. Secur. (MINES)*, 2010, pp. 851–855.

[24] L. Zhu and B. Tung, *Public Key Cryptography for Initial Authentication in Kerberos (PKINIT)*, document RFC-4556, 2006. Accessed: Feb. 10, 2017. [Online]. Available: https://tools.ietf.org/pdf/rfc4556.pdf

[25] G. Wettstein, J. Grosen, and E. Rodriquez, "IDfusion an open-architecture for Kerberos based authorization," in *Proc. AFS Kerberos Best Practices Workshop*, Michigan, MI, USA, 2006, pp. 1–22.

[26] J. Astorga, E. Jacob, M. Huarte, and M. Higuero, "Ladon[1]: End-to-end authorisation support for resource-deprived environments," *IET Inf. Secur.*, vol. 6, no. 2, pp. 93–101, Jun. 2012.

[27] N. Itoi and P. Honeyman, "Smartcard integration with kerberos V5," in *Proc. USENIX Workshop Smartcard Technol.*, 1999, pp. 1–12.

[28] H.-Y. Chien, J.-K. Jan, and Y.-M. Tseng, "An efficient and practical solution to remote authentication: Smart card," *Comput. Secur.*, vol. 21, no. 4, pp. 372–375, 2002.

[29] X. Li, J. Niu, M. K. Khan, and J. Liao, "An enhanced smart card based remote user password authentication scheme," *J. Netw. Comput. Appl.*, vol. 36, no. 5, pp. 1365–1371, 2013.

[30] J. Kohl and C. Neuman, *The Kerberos Network Authentication Service (V5)*, document RFC 1510, 1993. Accessed: Feb. 10, 2017. [Online]. Available: https://tools.ietf.org/pdf/rfc1510.pdf

[31] I. Downnard, "Public-key cryptography extensions into kerberos," *IEEE Potentials*, vol. 21, no. 5, pp. 30–34, Dec. 2002.

[32] N. Sakimura, J. Bradley, M. Jones, B. de Medeiros, and C. Mortimore, "OpenID Connect Core 1.0 incorporating errata set 1," OpenID Foundation, Tech. Rep., Nov. 2014. [Online]. Available: https://openid.net/specs/openid-connect-core-1_0.html

[33] G. S. Sadasivam, K. A. Kumari, and S. Rubika, "A novel authentication service for Hadoop in cloud environment," in *Proc. Int. Conf. Cloud Comput. Emerg. Markets (CCEM)*, 2012, pp. 1–6.

[34] S. M. Bellovin and M. Merritt, "Limitations of the Kerberos authentication system," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 20, no. 5, pp. 119–132, 1990.

[35] Z. Xu and K. M. Martin, "A practical deployment framework for use of attribute-based encryption in data protection," in *Proc. 10th Int. Conf. High Perform. Comput. Commun.*, 2013, pp. 1593–1598.

[36] J. C. Cohen and S. Acharya, "Incorporating hardware trust mechanisms in Apache Hadoop: To improve the integrity and confidentiality of data in a distributed Apache Hadoop file system: An information technology infrastructure and software approach," in *Proc. Globecom Workshops*, Dec. 2012, pp. 769–774.

[37] B. R. Chang, H. F. Tsai, Z.-Y. Lin, and C.-M. Chen, "Access security on cloud computing implemented in Hadoop system," in *Proc. 5th Int. Conf. Genet. Evol. Comput. (ICGEC)*, 2011, pp. 77–80.

[38] K. Hwang and D. Li, "Trusted cloud computing with secure resources and data coloring," *IEEE Internet Comput.*, vol. 14, no. 5, pp. 14–22, Sep./Oct. 2010.

[39] D. Yuefa, W. Bo, G. Yaqiang, Z. Quan, and T. Chaojing, "Data security model for cloud computing," in *Proc. Int. Workshop Inf. Secur. Appl. (IWISA)*, 2009, pp. 141–144.

[40] V. N. Inukollu, S. Arsi, and S. R. Ravuri, "Security issues associated with big data in cloud computing," *Int. J. Netw. Secur. Appl.*, vol. 6, no. 3, pp. 45–56, 2014.

[41] M. R. Jam, L. M. Khanli, M. S. Javan, and M. K. Akbari, "A survey on security of Hadoop," in *Proc. 4th Int. Conf. Comput. Knowl. Eng. (ICCKE)*, Oct. 2014, pp. 716–721.

[42] B. Campbell, C. Mortimore, and M. Jones, *Security Assertion Markup Language (SAML) 2.0 Profile for OAuth 2.0 Client Authentication and Authorization Grants*, document RFC 7522, 2015, Accessed: Feb. 10, 2017.

[43] Y. Kirsal and O. Gemikonakli, "Further improvements to the kerberos timed authentication protocol," in *Proc. Novel Algorithms Techn. Telecommun., Automat. Ind. Electron.*, 2008, pp. 550–554.

[44] N. T. Abdelmajid, M. A. Hossain, S. Shepherd, and K. Mahmoud, "Location-based kerberos authentication protocol," in *Proc. 2nd Int. Conf. Social Comput. (SocialCom)*, 2010, pp. 1099–1104.

[45] A. Joux, "The weil and tate pairings as building blocks for public key cryptosystems," in *Proc. Int. Algorithmic Number Theory Symp.*, 2002, pp. 20–32.

[46] S. Kalra and S. K. Sood, "Secure authentication scheme for IoT and cloud servers," *Pervasive Mobile Comput.*, vol. 24, pp. 210–223, Dec. 2015.

[47] S. Kumari, M. Karuppiah, A. K. Das, X. Li, F. Wu, and N. Kumar, "A secure authentication scheme based on elliptic curve cryptography for IoT and cloud servers," *J. Supercomput.*, pp. 1–26, 2017, doi: 10.1007/s11227-017-2048-0.

[48] J. H. Yang and P. Y. Lin, "An ID-based user authentication scheme for cloud computing," in *Proc. 10th Int. Conf. Intell. Inf. Hiding Multimedia Signal Process. (IIH-MSP)*, 2014, pp. 98–101.

[49] T.-H. Chen, H.-L. Yeh, and W.-K. Shih, "An advanced ECC dynamic ID-based remote mutual authentication scheme for cloud computing," in *Proc. 5th FTRA Int. Conf. Multimedia Ubiquitous Eng. (MUE)*, Jun. 2011, pp. 155–159.
[50] D. Wang, Y. Mei, C.-G. Ma, and Z.-S. Cui, "Comments on an advanced dynamic ID-based authentication scheme for cloud computing," in *Proc. Int. Conf. Web Inf. Syst. Mining*, 2012, pp. 246–253.
[51] Z. Hao, S. Zhong, and N. Yu, "A time-bound ticket-based mutual authentication scheme for cloud computing," *Int. J. Comput., Commun. Control*, vol. 6, no. 2, pp. 227–235, 2011.
[52] C. D. Jaidhar, "Enhanced mutual authentication scheme for cloud architecture," in *Proc. 3rd Int. Adv. Comput. Conf. (IACC)*, 2013, pp. 70–75.
[53] P. Gope and A. K. Das, "Robust anonymous mutual authentication scheme forn-times ubiquitous mobile cloud computing services," *IEEE Internet Things J.*, vol. 4, no. 5, pp. 1764–1772, Oct. 2017.
[54] J.-L. Tsai and N.-W. Lo, "A privacy-aware authentication scheme for distributed mobile cloud computing services," *IEEE Syst. J.*, vol. 9, no. 3, pp. 805–815, Sep. 2015.
[55] E.-J. Yoon and K.-Y. Yoo, "Robust biometrics-based multi-server authentication with key agreement scheme for smart cards on elliptic curve cryptosystem," *J. Supercomput.*, vol. 63, no. 1, pp. 235–255, 2013.
[56] D. Mishra, A. K. Das, and S. Mukhopadhyay, "A secure user anonymity-preserving biometric-based multi-server authenticated key agreement scheme using smart cards," *Expert Syst. Appl.*, vol. 41, no. 18, pp. 8129–8143, 2014.
[57] H. Shen, C. Gao, D. He, and L. Wu, "New biometrics-based authentication scheme for multi-server environment in critical systems," *J. Ambient Intell. Humanized Comput.*, vol. 6, no. 6, pp. 825–834, 2015.
[58] F. Wu, L. Xu, S. Kumari, and X. Li, "A novel and provably secure biometrics-based three-factor remote authentication scheme for mobile client–server networks," *Comput. Electr. Eng.*, vol. 45, pp. 274–285, Jul. 2015.
[59] D. He and D. Wang, "Robust biometrics-based authentication scheme for multiserver environment," *IEEE Syst. J.*, vol. 9, no. 3, pp. 816–823, Sep. 2015.
[60] V. Odelu, A. K. Das, and A. Goswami, "A secure biometrics-based multi-server authentication protocol using smart cards," *IEEE Trans. Inf. Forensics Security*, vol. 10, no. 9, pp. 1953–1966, Sep. 2015.
[61] M. Wazid, A. K. Das, V. Odelu, N. Kumar, and W. Susilo, "Secure remote user authenticated key establishment protocol for smart home environment," *IEEE Trans. Dependable Secure Comput.*, Oct. 2017, doi: 10.1109/TDSC.2017.2764083.
[62] S. Kumari, X. Li, F. Wu, A. K. Das, K.-K. R. Choo, and J. Shen, "Design of a provably secure biometrics-based multi-cloud-server authentication scheme," *Future Gener. Comput. Syst.*, vol. 68, pp. 320–330, Mar. 2017.
[63] J. Srinivas, A. K. Das, M. Wazid, and N. Kumar, "Anonymous lightweight chaotic map-based authenticated key agreement protocol for industrial Internet of Things," *IEEE Trans. Dependable Secure Comput.*, Jul. 2018, doi: 10.1109/TDSC.2018.2857811.
[64] Y. Yang, R. H. Deng, and F. Bao, "A practical password-based two-server authentication and key exchange system," *IEEE Trans. Dependable Secure Comput.*, vol. 3, no. 2, pp. 105–114, Apr. 2006.
[65] H. Jin, D. S. Wong, and Y. Xu, "An efficient password-only two-server authenticated key exchange system," in *Proc. Int. Conf. Inf. Commun. Secur.*, 2007, pp. 44–56.
[66] J. Katz, P. MacKenzie, G. Taban, and V. Gligor, "Two-server password-only authenticated key exchange," *Comput. Syst. Sci.*, vol. 78, no. 2, pp. 651–669, 2012.
[67] X. Yi, S. Ling, and H. Wang, "Efficient two-server password-only authenticated key exchange," *IEEE Trans. Parallel Distrib. Syst.*, vol. 24, no. 9, pp. 1773–1782, Sep. 2013.
[68] X. Yi, F. Hao, and E. Bertino, "Id-based two-server password-authenticated key exchange," in *Proc. Eur. Symp. Res. Comput. Secur.*, 2014, pp. 257–276.
[69] P. Sarkar, "A simple and generic construction of authenticated encryption with associated data," *ACM Trans. Inf. Syst. Secur.*, vol. 13, no. 4, 2010, Art. no. 33.
[70] *Federal Information Processing Standards Publication: Secure Hash Standard*, Standard FIPS PUB 180-1, National Institute Standards Technology, U.S. Department Commerce, Apr. 1995. Accessed: Sep. 2017. [Online]. Available: http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf
[71] R. W. D. Nickalls, "A new approach to solving the cubic: Cardan's solution revealed," *Math. Gazette*, vol. 77, no. 480, pp. 354–359, 1993.
[72] N. Koblitz, "Elliptic curve cryptosystems," *Math. Comput.*, vol. 48, no. 177, pp. 203–209, 1987.
[73] D. Dolev and A. C. Yao, "On the security of public key protocols," *IEEE Trans. Inf. Theory*, vol. 29, no. 2, pp. 198–208, Mar. 1983.
[74] R. Canetti and H. Krawczyk, "Analysis of key-exchange protocols and their use for building secure channels," in *Proc. Int. Conf. Theory Appl. Cryptograph. Techn.*, 2001, pp. 453–474.
[75] C. J. F. Cremers, "Formally and practically relating the CK, CK-HMQV, and eCK security models for authenticated key exchange," IACR Cryptol. ePrint Arch., Tech. Rep. 2009/253, 2009.
[76] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, 2008.
[77] M. Abdalla, P. Fouque, and D. Pointcheval, "Password-based authenticated key exchange in the three-party setting," in *Proc. 8th Int. Workshop Theory Pract. Public Key Cryptogr. (PKC)*, in Lecture Notes in Computer Science, vol. 3386. Berlin, Germany: Springer, 2005, pp. 65–84.
[78] C.-C. Chang and H.-D. Le, "A provably secure, efficient, and flexible authentication scheme for ad hoc wireless sensor networks," *IEEE Trans. Wireless Commun.*, vol. 15, no. 1, pp. 357–366, Jan. 2016.
[79] A. Armando *et al.*, "The AVISPA tool for the automated validation of Internet security protocols and applications," in *Proc. Int. Conf. Comput. Aided Verification*, 2005, pp. 281–285.
[80] L. Viganò, "Automated security protocol analysis with the AVISPA tool," *Electron. Notes Theor. Comput. Sci.*, vol. 155, pp. 61–86, May 2006, doi: 10.1016/j.entcs.2005.11.052.
[81] D. von Oheimb, "The high-level protocol specification language HLPSL developed in the EU project AVISPA," in *Proc. APPSEM Workshop*, 2005, pp. 1–17.

DURBADAL CHATTARAJ (S'17) received the B.Tech. and M.Tech. degrees in computer science and engineering from the West Bengal University of Technology. He is currently pursuing the Ph.D. degree with the Subir Chowdhury School of Quality and Reliability, IIT Kharagpur, Kharagpur, India. His current research interests include cloud data security and reliability, cryptography, cloud computing, big data analytics, information security, and dependable computing. He has authored seven papers in international journals and conferences in the above-mentioned areas.

MONALISA SARMA received the B.Tech. degree from the North Eastern Regional Institute of Science and Technology, Itanagar, in 1994, and the M.S. and Ph.D. degrees from IIT Kharagpur, Kharagpur, in 2003 and 2008, respectively, in computer science and engineering. She is currently an Assistant Professor with the Subir Chowdhury School of Quality and Reliability, IIT Kharagpur, India. She has about five years of teaching experience and four years of industrial experience (two years in Oil India Ltd. and two years in Siemens Corporate Technology, India). Her areas of interest and research include biometric security, cloud computing security, and dependability analysis.

ASHOK KUMAR DAS (M'17–SM'18) received the Ph.D. degree in computer science and engineering, the M.Tech. degree in computer science and data processing, and the M.Sc. degree in mathematics from IIT Kharagpur, India. He is currently an Associate Professor with the Center for Security, Theory and Algorithmic Research, International Institute of Information Technology, Hyderabad, India. His current research interests include cryptography, wireless sensor network security, hierarchical access control, security in vehicular ad hoc networks, smart grid, Internet of Things (IoT), cyber-physical systems, and cloud computing, and remote user authentication. He has authored over 175 papers in international journals and conferences in the above areas, including over 150 reputed journal papers. Some of his research findings are published in top cited journals, such as the IEEE TRANSACTIONS ON INFORMATION FORENSICS AND SECURITY, the IEEE TRANSACTIONS ON DEPENDABLE AND SECURE COMPUTING, the IEEE TRANSACTIONS ON SMART GRID, the IEEE INTERNET OF THINGS JOURNAL, the IEEE TRANSACTIONS ON INDUSTRIAL INFORMATICS, the IEEE TRANSACTIONS ON VEHICULAR TECHNOLOGY, the IEEE TRANSACTIONS ON CONSUMER ELECTRONICS, the IEEE JOURNAL OF BIOMEDICAL AND HEALTH INFORMATICS (formerly the IEEE TRANSACTIONS ON INFORMATION TECHNOLOGY IN BIOMEDICINE), the *IEEE Consumer Electronics Magazine*, the IEEE ACCESS, the *IEEE Communications Magazine*, *Future Generation Computer Systems*, *Computers & Electrical Engineering*, *Computer Methods and Programs in Biomedicine*, *Computer Standards & Interfaces*, *Computer Networks*, *Expert Systems with Applications*, and the *Journal of Network and Computer Applications*. He has served as a Program Committee Member in many international conferences. He was a recipient of the Institute Silver Medal from IIT Kharagpur. He is on the Editorial Board of the *KSII Transactions on Internet and Information Systems*, the *International Journal of Internet Technology and Secured Transactions* (Inderscience), and *Recent Advances in Communications and Networking Technology* and is a Guest Editor of *Computers & Electrical Engineering* (Elsevier) for the special issue on big data and IoT in e-healthcare.

NEERAJ KUMAR (M'16–SM'17) received the Ph.D. degree in computer science and engineering from Shri Mata Vaishno Devi University, Katra, India, in 2009. He was a Post-Doctoral Research Fellow with Coventry University, Coventry, U.K. He is currently an Associate Professor with the Department of Computer Science and Engineering, Thapar University, Patiala, India. He has authored more than 200 technical research papers published in leading journals and conferences from the IEEE, Elsevier, Springer, and John Wiley. Some of his research findings are published in top cited journals, such as the IEEE TRANSACTIONS ON INDUSTRIAL ELECTRONICS, the IEEE TRANSACTIONS ON DEPENDABLE AND SECURE COMPUTING, the IEEE TRANSACTIONS ON INTELLIGENT TRANSPORTATION SYSTEMS, the IEEE TRANSACTIONS ON CONSUMER ELECTRONICS, the *IEEE Network*, the *IEEE Communications*, the IEEE WIRELESS COMMUNICATIONS, the IEEE INTERNET OF THINGS JOURNAL, and the IEEE SYSTEMS JOURNAL. He has guided many research scholars leading to Ph.D. and M.E./M.Tech. He is on the Editorial Board of the *IEEE Communications Magazine*, the *Journal of Network and Computer Applications* (Elsevier), and the *International Journal of Communication Systems* (Wiley).

JOEL J. P. C. RODRIGUES (S'01–M'06–SM'06) received the five-year B.Sc. degree (licentiate) in informatics engineering from the University of Coimbra, Portugal, the M.Sc. degree and the Ph.D. degree in informatics engineering from UBI, Portugal, and the Habilitation degree in computer science and engineering from the University of Haute Alsace, France. He has been a Professor at UBI and a Visiting Professor at UNIFOR. He is currently a Professor with the National Institute of Telecommunications, Brazil, and a Senior Researcher at the Instituto de Telecomunicações, Portugal. He has authored or co-authored over 650 papers in refereed international journals and conferences and three books. He holds two patents. He is a member of the Internet Society, an IARIA Fellow, and a Senior Member of ACM. He received the Academic Title of Aggregated Professor in informatics engineering from UBI. He is the Leader of the Net-GNA Research Group, the President of the Scientific Council at ParkUrbis—Covilhã Science and Technology Park, the Past Chair of the IEEE ComSoc TCs on eHealth and on Communications Software, and a Steering Committee member of the IEEE Life Sciences Technical Community. He is the Editor-in-Chief of the *International Journal on E-Health and Medical Communications* and an Editorial Board Member of several journals.

YOUNGHO PARK (M'17) received the B.S., M.S., and Ph.D degrees in electronic engineering, Kyungpook National University, Daegu, South Korea, in 1989,1991, and 1995, respectively. From 1996 to 2008, he was a Professor with the School of Electronics and Electrical Engineering, Sangju National University, South Korea. From 2003 to 2004, he was a Visiting Scholar with the School of Electrical Engineering and Computer Science, Oregon State University, USA. He is currently a Professor with the School of Electronics Engineering, Kyungpook National University. His research interests include computer networks, multimedia, and information security.